

**Ege Üniversitesi**  
**Bilgisayar Mühendisliği**

**Model Tabanlı Geliştirme ve Mimari**  
**(İnceleme/Analiz Çalışması)**

**Ders**

**Model Tabanlı Yazılım Mühendisliği**

**Prof Dr. Yasemin TOPALOĞLU**

**Hazırlayan**

**Göksel ÜÇER**

## İçindekiler

<b>Şekiller</b> .....	<b>3</b>
<b>Önemli kısaltmalar</b> .....	<b>4</b>
<b>Giriş</b> .....	<b>5</b>
Temel Kavramlar .....	5
Domain .....	5
Metamodel .....	5
Model .....	5
Domain Specific Language (DSL) .....	6
MDD yönteminin faydaları .....	6
<b>Platform Bağımsız Modeller (PIM)</b> .....	<b>Error! Bookmark not defined.</b>
Computational independent model (CIM) .....	7
Platform independent model (PIM) .....	7
Platform specific model (PSM) .....	8
<b>MDA için Mimari Katmanlar</b> .....	<b>9</b>
M3 – Meta MetaModel .....	9
M2 – MetaModel .....	9
M1 – Model .....	9
M0 – Gerçeklik .....	9
<b>MDA – Metamodel Kullanımı</b> .....	<b>11</b>
UML .....	11
UML Profilleri .....	11
UML Profil Araçları .....	12
<b>Model Dönüşümleri</b> .....	<b>14</b>
Modelden Kod Üretmek .....	14
<b>Model Driven Software Development (MDSO)</b> .....	<b>16</b>
CASE ile karşılaştırma .....	16
Çevik süreçlerle karşılaştırma .....	16
MDSO Alanındaki Önemli İçerikler .....	16
<b>Sonuç</b> .....	<b>17</b>
<b>Kaynaklar</b> .....	<b>18</b>

## Şekiller

ŞEKİL 1 - CIM İÇİN ÖRNEK SATIŞ SÜRECİ .....	7
ŞEKİL 2 - PIM İÇİN ÖRNEK UML SINIF DİYAGRAMI.....	8
ŞEKİL 3 - MODEL DÖNÜŞÜMÜ .....	8
ŞEKİL 4 - MDA KATMANLARI .....	9
ŞEKİL 5 - UML PROFİL ELEMANLARI.....	12
ŞEKİL 6 - CORBA İÇİN ALT ALANLAR.....	12
ŞEKİL 7 - MODELLER ARASI ÖRNEK DÖNÜŞÜM.....	14

## Önemli kısaltmalar

- EAI Enterprise Application Integration
- CIM Computational Independent Model
- CORBA Common Object Request Broker Architecture
- DSL Domain Specific Language
- MDA Model Driven Architecture
- MDD Model Driven Development
- MDSD Model Driven Software Development
- MOF Meta Object Facility
- OMG Object Management Group
- QoS Quality Of Service
- PIM Platform Independent Model
- PSM Platform Specific Model
- XMI XML Metadata Interchange
- UML Unified Modeling Language

## Giriş

Bilgisayar üzerinde yapılan çalışmaların ve tasarımların sayısı gün geçtikçe artmakta ve buna paralel olarak araştırmacılar, bilgisayar ile yapılan tasarımların soruna odaklı olmasını, bilgisayar kavramlarından soyutlanmasını ve mümkün olabilecek en üst seviyede çalışma alanını temsil edecek tasarımların ortaya çıkmasını sağlayacak yöntemleri araştırmaktadırlar. Bu araştırmaların başarılı olması durumunda, sorunu temsil eden kavramlar ve tasarımlar yüksek bir oranla çalışma alanını temsil edecek ve yeri geldiğinde, tasarım ya da model çalıştırılmasında, zeki, geliştirilmiş ve çalışma alanının dilini tanıyan dönüştürücü uygulamalarla başka alanlara ya da tasarımlara çıktı verebilecektir.

Yukarıda sözü edilen tasarımdaki soyutlanma fikrine, Fortran, C++, Pascal ve Basic gibi yazılım derleyicileri örnek gösterilebilir. Yazılım dili derleyicileri sayesinde, ilk bilgisayar tasarımındaki platforma özel komutlara ya da zor tasarım komutlarına artık ihtiyaç duyulmamaktadır. Dillerin gelişmesi, araştırmaların soyutlanma ve daha kolay tasarım yollarını bulması sonucunda günümüz tasarımları artık daha gerçekçi ve dünya diline yakın bir düzendedir.

Bu çalışmada, *Model Driven Development* ve *Model Driven Architecture* konuları incelenecektir. MDD yaklaşımı, tasarım anında sistem modelinin geliştirilmesi ve sonra gerçek gereksinimlere dönüştürülmesi olarak açıklanabilir.

## Temel Kavramlar

### Domain

Model tabanlı yazılım geliştirmenin ilk noktası, çalışma alanıdır (*domain*). Bu kelime, ilgili alanın sınırları şeklinde düşünülebilir. Yazılım geliştirmede, bir problem üzerinde çalışma yapılacaksa ve bu probleme ait uzman kişilerin ya da bilgi tabanının olmasında fayda vardır. Bu faydaya ek olarak, çalışma alanının sınırlarının belirlenmesi, bu alana ait terminolojilerin tanımlanması ve hatta alanın büyük olması söz konusu ise alt alanlara bölünerek karmaşık bir problemin, basit anlaşılabilir sistemler haline getirilmesi gerekecektir.

### Metamodel

Metamodel kavramı, kullanılacak veya tasarlanacak modelin özelliklerini, yeteneklerini ve kurallarını tanımlamaktadır. Eğer bir çalışmada kullanılan modelin, geçerliliğini kontrol etme veya başka bir modele aktarma gereksinimi ortaya çıkarsa metamodel'in kullanımı yeterli olacaktır. Metamodel ifadelerinde, dilin durağan anlamsallığı (*static semantics*) ve soyut söz dizimi (*abstract syntax*) yer almaktadır. Bu iki kavramın yer almasıyla, tasarım anında bir çok kontrol yapılabilir ve metamodelle ait modellerin geçerliliği kontrol edilebilir.

### Model

Model kavramı MDA alanında önemli bir yer tutmaktadır. Model kavramını, iki ana başlık halinde açıklayabiliriz.

1. Gerçek sisteminin basitleştirilmiş görünümüdür (Selic, 2003)
2. Çalışma anında sistemi ifade edebilecek tanımlamalar kümesidir. (Seidewitz, 2003)

## **Modelin oluşturulması aşamasında aşağıdaki maddelere dikkat edilmelidir.**

- Gerçek sistemin tam olarak nasıl ifade edildiği açık şekilde gösterilmelidir ve anlaşılabilirlik oranı en üst seviyede tutulmalıdır.
- Sistemi anlatmak ve ifade etmek için kullanılan modelin, bazı özel durumları anlatamaması durumu söz konusu ise, özel işaretler ile ek tanımlama yapılmalıdır.
- Model mutlaka gerçek yaşam özelliklerine ve beklentilerine göre tasarlanmalıdır.
- Tasarlanan modeller denemelere açık olmalıdır, değişik sonuçları ortaya çıkarabilmelidir ve/veya bilgisayar ortamında çalıştırılarak test edilebilmelidir.
- Model tasarımında mutlaka ucuz yöntemler kullanılmalıdır. Gerçek sistem modelini kurmak için kullanılacak yöntemin pahalı ve zamanı alıcı olmamalıdır.

### **Domain Specific Language (DSL)**

DSL, bir problem alanının standart kelimelerini kullanarak çözümün ifade edilmesidir. Standart terminolojinin kullanılması, karmaşık problemlerin çözümünde her zaman yeterli olmaz. Buna ek olarak mutlaka yüksek seviyede soyutlanmış modellerin de görev alması gerekmektedir. MDSK kapsamında yapılan çalışmaların en son durağı DSL olarak düşünülebilir. Her problem, bir model ile ifade edilebilir, soyut kavramlarla çözüme yaklaşılabılır ve sonunda görev alacak platforma dönüştürülebilir.

### **MDD yönteminin faydaları**

Yazılım alanında MDD'in kullanılması, tasarımın öncesinde (*ön analiz aşaması, gereksinim analizi*), tasarım anında (*yazılım aşaması, kodlama*), test ve / veya bakım & onarım aşamasında tasarımcıya olumlu katkılar sağlayabilir.

- Karmaşık sorunların daha kolay anlaşılmasını sağlar.
- Ürünün ve tasarımın, bilgisayar platformundan bağımsız şekilde tasarlanması sonucunda, yürütülen projeye, uzmanlar daha etkin katkı sağlayabilir. Sonuç olarak yazılım ihtiyaçları daha doğru ve gerçekçi tespit edilebilir.
- Yazılım evlerinde, koda ve karmaşık algoritmalara bağlı olarak geliştirilen ürünlerin zamanla artması, ortaya bir risk çıkarmaktadır. Bu riski en az seviyede tutmak için belgeleme ve yazılım sürümlendirme yöntemleri kullanılmaktadır ancak zamanla bu yöntemlerin yoğun kullanılmasına rağmen yazılımda yapılacak değişikliklerde zorluklar ortaya çıkması kaçınılmazdır. MDD ile yazılımdaki ihtiyaçların model olarak gösterilmesi ve sunulması, yapılacak değişikliğin daha risksiz gerçekleştirilmesine katkı sağlayacaktır.
- Bilgisayar teknolojilerinin hızla gelişmesi ve rekabet ortamında günden güne yeni kavramların keşfedilmesi sonucunda değişik amaçlara hizmet eden *framework*'ler ortaya çıkmaktadır. Ürün ilk başta sabit bir platform üzerinde tasarlanabilir ancak sonra başka bir platforma geçiş istenirse, MDD'nin modelleri ve bu modellere ait dönüşüm araçları sayesinde geçiş maliyeti daha az ve geçiş kısa sürede olacaktır.

## Platform

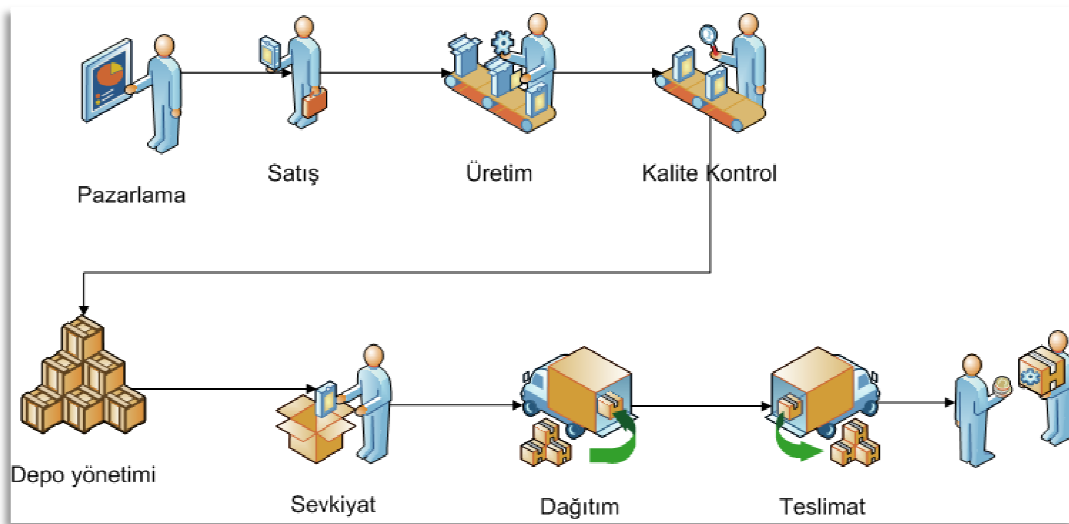
Tasarımcılar ürün geliştirirken donanım, işletim sistemi, işlemci, ağ performansı ve uygulamalar arası bütünleşik çalışma zorunluluklarından bağımsız tasarım yapmalıdırlar. Bağımsız şekilde yapılacak tasarımlarda soyutlanma üst seviyede olacak, gerçek ihtiyaçlar ve gereksinimler daha fazla karşılanacak, müşteri ile ortak çalışma sırasında olabilecek değişiklikler veya test sırasında ortaya çıkabilecek eksiklikler daha etkin şekilde yönetilebilecektir.

Platform bağımsız tasarımın faydalarına, tek bir platformda tasarım yaparak da erişilebilir ancak günümüz teknolojisiyle ve sunulan çeşitli zengin platformlarla bu yöntemi kullanmak çok akıllıca olmayabilir. Yazılım dünyasını örnek alırsak, Microsoft .Net framework ve Java Virtual Machine (JVM) platformları, soyut sınıfları ve *component* servisleri (*WCF & Remoting, J2EE & Enterprise Service vb*) ile tasarımcıların ürün geliştirme sürelerini daha hızlı ve verimli hale getirmektedir. Ancak bu teknolojilerden birini seçmek ve sunulan özelliklere göre tasarım yapmak, yazılım sistemini ve tasarımı platform bağımlı bir hale getirecektir.

MDA, sistem tasarımında ve analizinde; CIM, PIM ve PSM kavramlarını tanımlamaktadır.

## Computational independent model (CIM)

CIM, bilgisayar kavramlarından arındırılmış, sistemin yapısal detayını gizleyen ancak şekil 1'de gösterildiği gibi sisteminin bütününe anlaşılmasını kolaylaştıran modeldir.

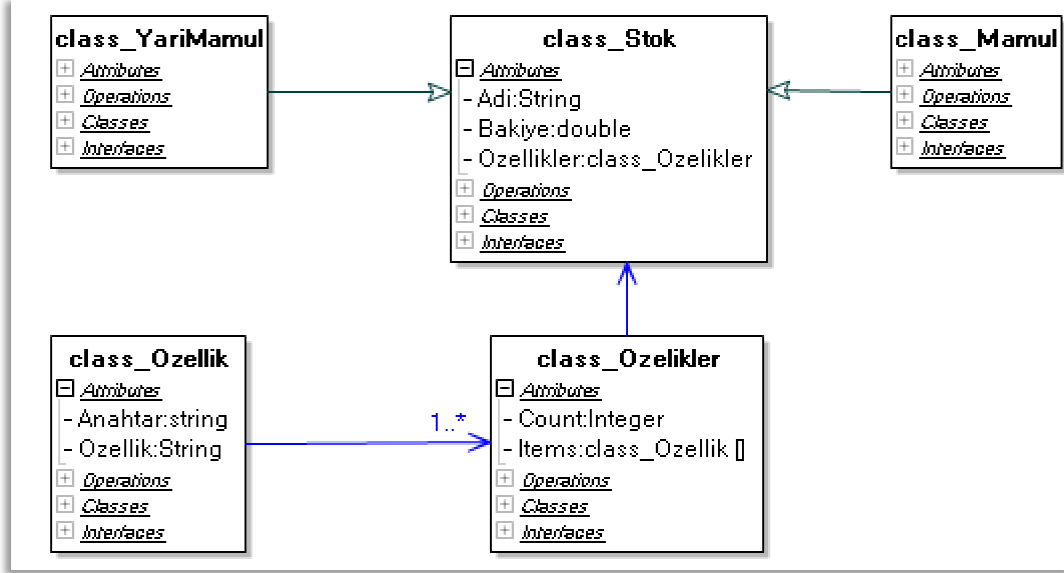


Şekil 1 - CIM için örnek satış süreci

## Platform independent model (PIM)

PIM, bilgisayar kavramları ile tasarlanmış modeldir ancak özellikle bir bilgisayar platformunu ya da teknolojisini tanımlamaz. PIM bilgisayar platformlarının üst seviyesinde kullanılan model olarak düşünülmelidir.

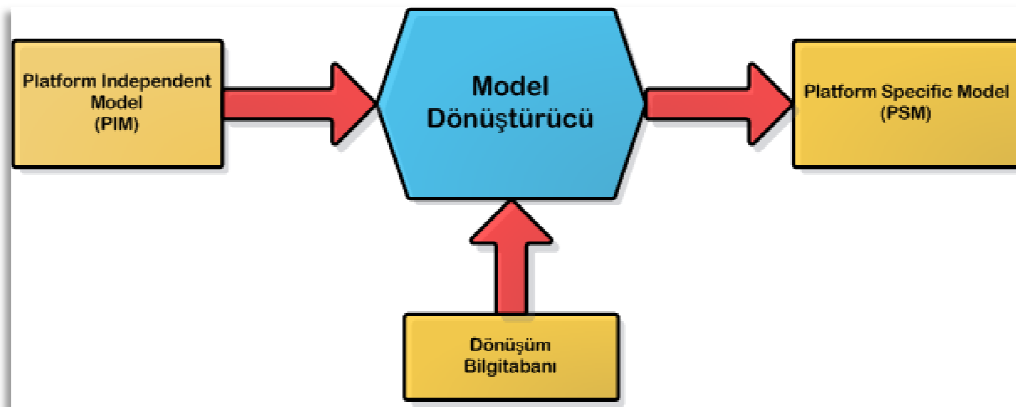
UML diyagramları ile temsil edilen bir yazılım sistem analizi, PIM için örnek gösterilebilir. Analizde yer alacak UML sınıf diyagramları platform bağımsız düşünülebilir. Bu modeller, daha sonra, ihtiyaç olması durumunda özel bir platforma( Java veya C# gibi yazılım geliştirme platformlarına) özel uygulamalarla aktarılabilir.



Şekil 2 - PIM için örnek UML sınıf diyagramı

### Platform specific model (PSM)

PSM, bilgisayar sistemlerine göre özelleştirilmiş modeldir. MDA sürecindeki temel amaç, tasarımcıları PSM seviyesinden PIM ve CIM seviyesine kaydırmaktadır. Üst seviyede, başka bir ifadeyle soyut seviyede, tasarım tamamlandıktan sonra şekil 3'te gösterilen yöntemle, yetenekli ve zeki araçlar kullanılmalıdır ve kaynak modelden hedef modele dönüşüm yapılmalıdır. Bu dönüşümün sonrasında PSM seviyesinde çalıştırılabilir ve platforma göre özelleştirilmiş kod ya da tasarım parçaları kullanılabilir.



Şekil 3 - Model Dönüşümü

## MDA için Mimari Katmanlar

MDA, şekil 4'te gösterilen mimariye ve birkaç önemli standarda göre tasarlanmıştır. Bu standartlar aşağıdaki gibidir;

- Meta Object Facility (MOF)
- Unified Modeling Language (UML)
- XML Metadata Interchange (XMI)

### M3 – Meta MetaModel

Bu seviye, soyut, kendi kendini ifade edebilen ve metamodel'leri yöneten katmandır. UML veya MOF'inin kendisi bu seviyede tanımlanmaktadır.

### M2 – MetaModel

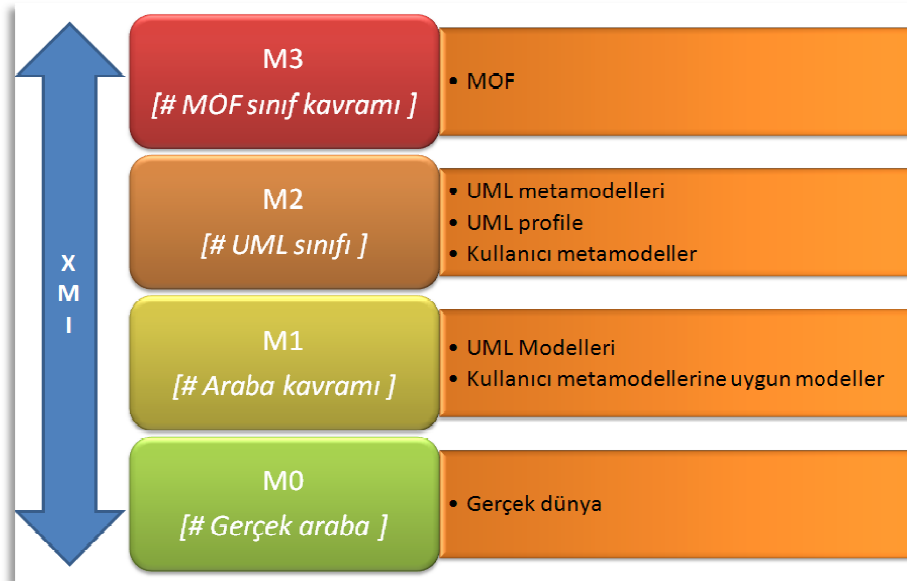
Standart tüm metamodeller bu seviyede yer almaktadır. UML Metamodel özellikleri, örnek olarak; sınıf özellikleri, sınıflar arasındaki ilişkiler, ilişki kurallarının hepsi UML metamodelinde tanımlanmıştır.

### M1 – Model

Gerçek dünyaya ait birçok kavram bu seviyede yer almaktadır. UML metamodeline uygun şekilde tasarlanmış birçok yazılım sisteminin UML modelleri, bu seviyede yer alanlara örnek gösterilebilir.

### M0 – Gerçeklik

M1 seviyesinde modellenen tüm tasarımlar bu katmanda gerçek dünya ile eşlenmektedir.



Şekil 4 - MDA katmanları

Araba kavramanın yazılım sistemi ile temsil edilmesi durumunda, yöntem olarak MDA kullanılırsa, katmanlar arası geçişlerde arabanın soyut dünyadan gerçek dünyaya geçişi şekilde 4'te [#] ifadeleri ile gösterilmiştir.

## MDA – Metamodel Kullanımı

### UML

UML, yazılım sistemlerinin belgelendirilmesi veya görsel hale getirilmesi için kullanılan dildir. Bu dilin kullanımı ile soyut modeller tasarlanabilir, karmaşık sistemler basitleştirilebilir, analiz edilebilir ve sonrasında yazılım geliştirme ortamlarına dönüştürülebilir.

**UML dili aşağıda gösterilen diyagramları kapsamaktadır;**

- Use case diagram
- Class diagram
- Behavior Diagram
  - Statechart diagram
  - Activity diagram
- Interaction diagram
  - Sequence diagram
  - Collaboration diagram
- Implementation diagram
  - Component diagram
  - Deployment diagram

### UML Profilleri

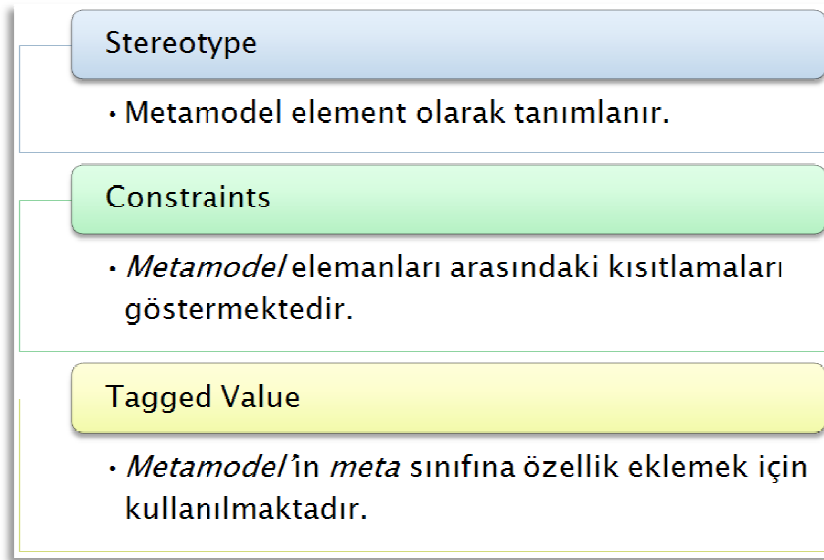
Model tabanlı geliştirmede, çalışma alanını temsil etmek için kullanılacak dil her zaman yeterli ya da tüm ifadeleri kapsayacak şekilde olmayabilir. Bu durumda, iki yöntem izlenerek çalışma alanının (*domain*) temsil yeteneği arttırabilir.

İlk yöntem olarak, MOF ailesine yeni bir dil destekleme seçeneği kullanılabilir. Ancak bu yöntemde MOF temel elemanlarını ve kavramları kullanarak yeni bir dil tasarlanacağı unutulmamalıdır. Bu durumda, hazır bir dil kullanılmadığı için tüm kavramları ifade edecek kurallar ve notasyonlar iyi tanımlanmalıdır. Çalışma alanı, bilgisayar dünyasına çok uzak değilse veya daha önceden bu alana ait bir temsil dili varsa, bu yöntemi kullanmak maliyetli, zor ve zaman alıcı olacaktır.

Diğer bir yöntem ise, UML profillerinin kullanımı ile ilgilidir. Buradaki temel amaç, mevcut UML dil yeteneklerini tekrar kullanmak ve çalışma alanına göre basit adımlarla özelleştirmektir. Özelleştirme yöntemi, mevcut nesnel tasarımlardaki genişletme tekniğine benzetilebilir. UML kavramlarını başka bir ifadeye çevirmek anlamsızdır ancak bu ifadeyi genişletmek mümkün olabilir.

UML profillerinde, mevcut UML ifadeleri ve kuralları kullanılmaktadır. Bu kavramlara değişik anlam ve özellikler şekil 5'te gösterilen elemanlarla, ilk temsil ettiği dünyayı bozmadığı sürece eklenebilir ya da çıkarılabilir. UML metamodelinde yer alan sınıf özelliğini örnek (*instance*), ilişki özelliğini de özellik (*attribute*) olarak göstermek söz konusu olamaz

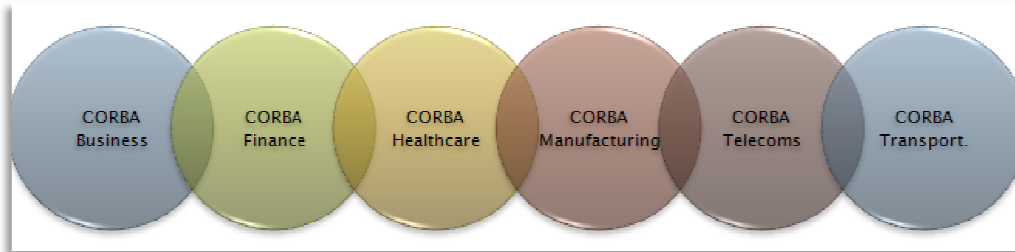
ancak ilişki üzerine sayısal ifadeler eklenebilir, özellik değerine ise o sahaya ait ek anlamlar yüklenebilir. (Tekil, sadece okunur ( *readonly* ), *array* vb.)



Şekil 5 - UML profil elemanları

OMG tarafından geliştirilen ve UML profil tekniği kullanılarak genişletilmiş UML profilleri aşağıdaki gibidir.

- Şekil 6'daki CORBA alt gruplarının tümünü temsil eden UML profili



Şekil 6 - CORBA için alt alanlar

- Enterprise application integration (EAI) için UML profili
- Quality Of Service için UML profili
- Ses tabanlı uygulamalar için UML profili

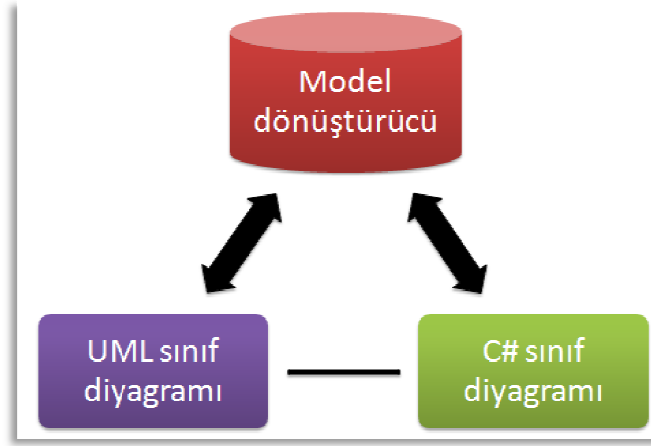
## UML Profil Araçları

UML profil yöntemi ile model genişletmek için yetenekli araçlara ihtiyaç vardır. Bu araçlarla, mevcut UML'i genişletilebilir, çalışma alanını daha iyi temsil edecek bir dil hazırlanabilir. Ortaya çıkacak dilin tasarımında, MOF metamodeline dolayısıyla da UML metamodeline bağlı kalmak ve daha önceden tanımlanmış kuralları bozmamak gerekir. Kuralların denetlendiği, UML profil tekniklerinin doğru uygulandığı ve dilin tasarlanacağı araçlar aşağıda listelenmiştir.

- *“IBM Rational UML Profiles”*,
- *Eclipse*
- *Borland Together Edition*
- *MagicDraw*

## Model Dönüşümleri

MDD ile hazırlanan modeller, ait oldukları metamodel kurallarını kullanarak başka modellere dönüştürülebilirler. Bu tür dönüşümleri yapabilmek için, şekil 7’de gösterilen örnek gibi, iki modele ve modeller arası dönüşüm işini yapacak ve yeterli bilgi tabanına sahip dönüşüm modeline ihtiyaç olacaktır.



Şekil 7 - Modeller arası örnek dönüşüm

Modeller arası dönüşümler tek yönlü veya çift yönlü olabilir. Tek yönlü model dönüşümlerinde, işlem sonrasında kaynak modele geri dönüş yapılamaz ancak çift yönlü model dönüşümler her ikisi de mümkün olacaktır.

Çift yönlü model dönüşümlerinin eksiksiz yapılabilmesi için her zaman model kavramları yeterli olmayabilir. Bu tür durumlarda, modelin izin verdiği kapsamda, etiketleme ifadeleri kullanılabilir. Bu ifadeler, kaynak modelde anlamı olmayan ancak dönüşüm sonrasında hedef modelde anlam kazanacak ifadeler olabilir. Modelde çift yönlü dönüşüm özelliğinin aktif olması durumunda ters mühendislik yöntemi ile modele geri dönüş yapılabilir.

## Modelden Kod Üretmek

Model tabanlı tasarımlarda, PSM seviyesine yaklaşmak, yazılım sistemleri için kod üretmek anlamına gelmektedir. Kod üretmek için, modeller arası dönüşümde de olduğu gibi, dönüşüm modeline ihtiyaç vardır. Kod üretiminde, geri dönüş yapılmayacaksa, kod söz dizimi için ek modele ihtiyaç olmadan dönüşüm yapılabilir( ve kod üretilebilir).

Modelden kod dönüşümü yapıldıktan sonra, özelleştirilecek ifadelerin toplam modele oranı %5 ile %15 arasında olmalıdır (Selic, 2003). Bu oran, teknolojideki gelişmelerle, model araçların daha yetenekli hale gelmesiyle, UML profilleriyle ve hedef modele dönüşümde anlamlı olacak makro ifadelerin kullanılmasıyla daha aşağıya çekilebilir.

Derleyicilerin çıktığı dönemde, yazılan satırların makine diline çevrilmesi ve insanlar tarafından kolay anlaşılabilir olmaması söz konusu olduğunda, bu yeni dilin, çevrim sonrasında güvenilirliği tartışılmıştır ancak daha sonra derleyicilerin ürettiği kod satırları yıllarca sorunsuz çalışmış ve güvensizlik ortamı artık yerini daha da ileri seviyedeki karmaşık

derleyicilere bırakmıştır. Günümüzde, modellerden kod üretimi yapılması durumunda benzer bir şüphe duyulabilir ancak modelin koda dönüşümü, test kodları ile kontrol altına alınırsa, çalışma alanına göre tasarlanan (*domain specific language – DSL*) dilin de kalitesi kontrol edilebilir.

## Model Driven Software Development (MDSO)

### CASE ile karşılaştırma

Bilgisayar destekli yazılım mühendisliği araçlarına bakıldığında, ilgili bir alan için özelleştirilmiş ve o alana ait özelliklerle donatılmış olduğu görünecektir. C# için geliştirilen bir dil geliştirme ortamı örnek alındığında, birçok özellik, menu, işlem ve *refactoring* gibi işlemlerin tamamen C# dilinin yeteneklerine, kurallarına ve özelliklerine göre davranacaktır. MDSO ise geliştirme ortamı olarak değil de yöntem olarak incelenmelidir. MDSO herhangi bir alan için özelleşebilir, o alanın terminolojisini kullanabilir ve yazılım geliştirme süreci o alanın uzmanları tarafından da desteklenerek sürdürülebilir.

### Çevik süreçlerle karşılaştırma

Çevik süreç bildirelerindeki en temel özellikleri sıraladığımızda;

- Basitleştirilmiş yazılım geliştirme sisteminin olmasını,
- Müşterilerin yazılım geliştirme sürecine dahil edilmesini,
- Anlaşılır ve basit belgelerle yazılım sürecinin sürdürülmesini,
- Ve küçük parçaları bir araya getirerek karmaşık sistemin tamamını oluşturmasını esas aldığımızı görmekteyiz.

MDSO tarafından bakıldığında, modelin çalışma alanını temsil edecek terminolojiyi kullanması, CIM ile bilgisayar mühendisliğinden uzak kavramlarla çözüm sağlanması ve modelin de iyi bir belgeleme çıktısı üretmesi düşünüldüğünde çevik yaklaşım süreçlerindeki bildirge maddelerine benzer olduğu görünecektir.

MDSO yönteminde, model çok önemlidir. Tüm problemler model tabanlı bir gösterimle çözümlenebilir ancak modelin sonunda çalıştırılacağı veya hizmete sunulacağı platform, tasarım anında göz ardı edilmemelidir ve aralarında mutlaka bir denge kurulmalıdır. *Domain* ve *Platform* kavramları mümkün olabildiğince birbirlerine yakın olmalıdır. Uzak olması durumunda yakınlaştıracak yöntemleri (*UML profiling gibi*) kullanmak faydalı olacaktır.

### MDSO Alanındaki Önemli İçerikler

- **Yazılım mimarisi**, yazılım sistemlerinin oluşturulması için yapısal kavramları tanımlar
- **Domain mimarisi**, yazılım mimarisi ile hemen hemen aynıdır ancak bu mimarinin içinde tüm sisteme ait metamodeller, DSL tanımlamaları, platformlar ve bunlar arasındaki dönüşümler yer almaktadır.

## Sonuç

Teknoloji, bilgisayar ve yazılım mühendisliği alanında gelişmeler hızla devam ediyor. Yazılım geliştirme süreci bu gelişime ayak uydurmakta ve çözülmesi gereken problemlerde yeni bir yöntem kullanmaktadır.

Bu yöntem ile tasarımcılar asıl soruna odaklanıyor, soyut kavramları bir araya getiriyor ve karmaşık sistemlerin çözülmesini sağlayabilirler.

Bu yöntemin başlıca elemanları arasında model ve platform kavramları yer almaktadır. Problemleri modellerle temsil etmek, edilememesi durumunda mevcut modelleri genişletmek ve çalışma alanına ürünleri gerçek dünya beklentileri ile tasarlamak tasarımcıların işlerini kolaylaştıracak ve daha yaratıcı, hatası az ürünlerin ortaya çıkmasına yardımcı olacaktır. Kalitesi yüksek, anlaşılabilir ve yeri geldiğinde başka modeller içinde çalıştırabilir modellerin son noktası ise platformdur.

Modelin platforma geçiş süreci, dönüştürücü uygulamaların kullanımı ile mümkün olabilir. Bu tür araçların yetenekleri ne kadar ilerlerse, model tabanlı geliştirme sürecinin kullanımını da o kadar arttıracaktır. MDD henüz çok yeni ve kullanımı az orandadır. Etkin ve çevik projelerde, müşterinin, başka bir ifadeyle, alanda uzman kişilerin görev almasının faydası kabul ediliyorsa, model tabanlı yazılım geliştirme de aynı şekilde düşünülebilir. Soruna odaklanma, değişime hızlı cevap vermek, platform bağımsızlığı ön planda tutmak yazılımın özelliklerinden ise, MDD kullanımı, süreci daha etkin hale getirecektir.

## **Kaynaklar**

Atkinson, C., “Model-driven development: A modeling foundation”, 2003

Fuantez, L., “An introduction to UML profiles”, 2004

Gasevic, D., Djuric, D., Devedzic Vladan, “Model driven architecture and ontology development”, Springer, 2006

OMG CORBA, “UML profile for CORBA, version 1.0”, 2002

OMG EAI, “UML profile for enterprise application integration (EAI)”, 2004

Seidewitz, E., “What model means”, 2003

Selic, B., “The pragmatics of model-driven development”, 2003