Yazılım Geliştirmede Anlam Sorunları ve Model Güdümlü Yaklaşım

Ahmet Egesoy[1] Yasemin Topaloğlu[2]

[1,2]Bilgisayar Mühendisliği Bölümü, Ege Üniversitesi, İzmir

[1]e-posta: ahmet.s@ege.edu.tr [2] e-posta: yasemin.topaloglu@ege.edu.tr

Özetçe

Yazılım geliştirme etkinlikleri, teknolojik ilerlemelerle birlikte donanım kaygılarından uzaklaşmakta ve anlamsal hedeflere yaklaşmaktadır. Bu gidişin en güncel kilometre taşlarından olan model güdümlü mühendislik (M.G.M.), modelleri yazılım geliştirme projelerinin öncelikli ürünleri olarak görmektedir. Model kavramının özgün bir gerçekleştirimine dayanması durumunda bu yaklaşımın yazılım geliştirme sürecindeki anlamsal eksikliği gidermesi mümkündür. Bu çalışmada nesneye yönelik yazılım geliştirme teknolojisindeki anlam tanımlama problemi tartışılmış ve anlam tanımlama konusunun M.G.M. açısından önemi incelenmiştir.

1. Giriş

Nesneye yönelik yazılım geliştirme paradigması, günümüzde yazılım mühendisliği alanındaki teknolojik standartların belirlenmesinde önemli bir paya sahiptir. Günümüz yazılım geliştirme anlayışının ana ekseni UML (Unified Modeling Language) modelleme dili [1] çevresinde şekillenmiş yöntem ve teknolojilere dayanmaktadır. Nesneye yönelik programlama yaklaşımı ve UML dili anlam tanımlama konusunda güçlü değildir ve bu eksiklik yazılım geliştirme sürecinin bazı aşamalarında tekrar eden sorunlar yaratmaktadır.

Yazılım geliştirme sürecinde, çözümleme aşamasından tasarım aşamasına geçişte karşılaşılan güçlüklerin bütünü *semantik boşluk* (anlambilimsel boşluk) olarak adlandırılmaktadır. Bu kavram dilbilim alanına aittir ve aynı nesnenin veya kavramın farklı dillerde ifade edilmesi konusunda yaşanan güçlüğü belirtmektedir. Ne var ki çözümleme aşamasından tasarım aşamasına geçiş, diller arasında çeviri yapmaya göre çok daha karmaşık bir işlemdir.

Anlam tanımlama sorununa bir çözüm olarak sunulan en önemli teknoloji OCL [2] (Object Constraint Language) adı verilen ve UML ile birlikte kullanılan bir metinsel dildir. OCL bir kısıt tanımlama dilidir ve sözdizim düzeyinden anlam düzeyine yapılması gereken atlamayı gerçekleştirmekten uzaktır. Ayrıca geliştiriciler arasında yaygınlaşmamıştır ve her ne kadar UML'in doğal bir uzantısı olduğu öne sürülse de [2] UML ile ilgili kitaplarda genellikle ya hiç söz edilmemekte [1] ya da fazla cesaret verici olmayan ifadelerle anlatılmaktadır [3].

Model Tabanlı Mühendislik (MDE: Model Driven Engineering) [4] Nesneye Yönelik Programlama Paradigması ve ona ilişkin yazılım geliştirme yöntem, araç ve standartlarından sonra gelen akımlar arasında gerçek bir anlamsal devrim yaratmak konusunda en umutlandırıcı olandır. Model kavramının ön plana çıkacağı bir paradigma geçişi sonrasında yazılım geliştirme çalışmalarına, nesnelerin bir işbölümü içinde birleştirilmesi anlayışından çok, uzman geliştiricilerin problem çözme deneyimlerini aktardıkları bazı hazır modeller üzerinde işlem ve dönüşümler gerçekleştirme anlayışının egemen olacağı düşünülmektedir [5, 6]. Kod odaklı bir yazılım geliştirme sürecinin yerini model odaklı bir yazılım geliştirme sürecinin alması, yazılımın kalitesinin ve kalıcılığının artırılmasına yönelik, arzu edilen devrimsel bir dönüşümdür [7].

Bu çalışma MGM'nin kendi başına bir paradigma olmasına yönelik daha önceki bir çalışmamızın [8] devamı niteliğindedir.

2. Sözdizim ve Anlam

Sözdizim bir dile ait örneklerin nasıl yaratılabileceğini veya doğrulanabileceğini gösteren bir bilgi kümesidir. Dolayısıyla sözdizim dilin geçerli olan elemanlarını ve bu elemanların geçerli olan bir araya geliş biçimlerini tanımlar. İsminin düşündürdüğünün tersine sözdizim kavramı sadece sözel veya metinsel bilgi ile ilgilenmez, görsel diller de dahil olmak üzere tüm dillerin birer sözdizimi vardır. Metinsel diller BNF ve EBNF gibi gramer tanımlama dilleri ile ifade edilirken, şekilsel diller için meta-modelleme veya çizge gramerleri gibi araçlar kullanılmaktadır.

Gramer kavramı ve BNF gibi diller, programlama dillerinin sözdizimini biçimsel olarak göstermeye yarayan araçlar olarak uzun zamandan beri etkili bir şekilde kullanılmaktadır. Anlam konusu ise sözdizime kıyasla oldukça sorunlu bir alandır. Her ne kadar işlevsel, gösterimsel (denotational) veya varsayımsal (axiomatic) anlambilim gibi yaklaşımlar sınırlı alanlarda ve karmaşık olmayan yazılımlar için iş görebilse de, gerçek uygulamalar için anlam konusunda biçimsel gösterim sağlamada pratik bir başarı elde etmiş herhangi bir yaklaşım henüz yoktur.

3. Alan Çözümlemesinde Anlam Sorunları

Yazılım geliştirme sürecinin ilk bölümünü oluşturan alan çözümlemesi safhası, çok sayıda bağlantı içeren gerçek dünyanın, yazılımın ilgi alanına bağlı olarak sınırlandırılmış bir görüntüsünü oluşturmaktadır. Gerçek dünyadan bilginin edinilmesi ve ifade edilmesi, kaçınılmaz olarak bilgi kaybı içeren bir işlemdir. Gerçek dünyada yer alan herhangi bir birim sayısız bağlantı ile evrenin geri kalanına bağlıdır ve bu bağlantıların her biri, birimin anlamına küçük bir katkıda bulunmaktadır. Modelleme araçları ise sınırlı bir kapasiteye sahiptir ve nesnelerin taşıdıkları anlamın tam olarak

kodlanması değil proje açısından önemli olan özelliklerinin ifade edilmesi söz konusudur.

Alan modelleri genellikle tasarım modelleri ile aynı dil kullanılarak ifade edilmektedir ve bu dil genellikle UML (Unified Modeling Language) [1] dilidir. UML dilinin sınıf diyagramları bir bilgisayar programının yapısının modellenmesinin yanı sıra, gerçek dünya kavramlarının modellenmesinde kullanılabilmektedir.

Bu kullanım pratik olsa da gerçekte çok sağlıklı bir seçim değildir. Örneğin çözümleme safhasında kullanılan kalıtım ilişkisi ile tasarım aşamasında kullanılan kalıtım ilişkisi anlam olarak birbirinden farklıdır. Çözümleme safhasında kullanılan kalıtım ilişkisi bir bilgi gösterimi sembolü olarak görülebilir. Tasarım dokümanları içinde kullanılan kalıtım ilişkisi ise tasarımın ve kodun yeniden kullanılabilirliği ile ilgilidir. Kalıtım hem kodu yeniden kullanılması daha kolay parçalara böler hem de kendisi bir yeniden kullanım biçimidir.

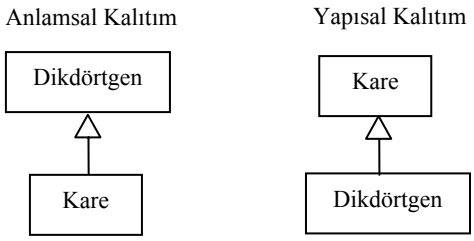
MDE biçimselleştirme ve otomasyon gibi hedeflere sahip olduğu için bir sistemi oluşturan parçalar arasındaki ilişkilere tamamen sembolik olarak yaklaşmak durumundadır. Bir ilişkiyi belirten sembolün süreç içinde iki farklı anlamda kullanılması geliştiricilerin dahi zaman zaman kafalarını karıştıran bir durumdur ve otomasyon hedefi güden MDE açısından açık bir sorun oluşturur.

Anlamsal tip ile sözdizimsel tip arasındaki fark, geometrik şekilleri gösteren sınıflar gibi çok boyutlu nesnelerde belirgin hale gelmektedir.

Kenar uzunluğunu gösteren *kenar* adında ve *tamsayı* tipinde tek bir özelliği olan bir *Kare* sınıfımız olduğunu varsayalım. Bir süre sonra daha karmaşık bir geometrik şekil olan *Dikdörtgen* için bir sınıfa ihtiyaç duyduğumuzda artık tek bir kenar uzunluğunun yeterli olmadığını ve *en* ve *boy* olarak iki ayrı kenar uzunluğun gerektiğini fark ederiz. Bu durumda *Kare* sınıfına ait kodun yeniden kullanılabilirliği üzerine fazla düşünmeden verilecek olan bir tasarım kararı *Dikdörtgen* sınıfının *Kare* sınıfından türetilmesi yönünde olabilir. *Dikdörtgen* sınıfı *Kare* sınıfını genişletirken *kenar2* isminde yeni bir özellik tanıtır ve var olan *kenar* özelliği ile birlikte dikdörtgenin iki kenar uzunluğu belirtilmiş olur.

Tecrübeli bir programcının yapmayacağını kolayca tahmin edebileceğimiz bu tür basit bir hata aslında yapısal düşünce tarzı açısından kendine özgü bir yerel tutarlılık içermektedir. Örnekte yeni bir sınıf yaratılmak istenmektedir (*Dikdörtgen* sınıfı) ve bu yeni sınıfın içeriğini oluşturan özelliklerin bir bölümü, zaten var olan bir sınıfın (*Kare* sınıfı) özelliklerinin tümünü içermektedir. Yeniden kullanımı amaç edinmiş (hayali) bir *yazılım geliştirme otomatı*, kalıtım kullanmaya karar vermek için iyi nedenlere sahiptir.

Diğer yandan yazılım geliştiriciler için *Kare*'den *Dikdörtgen*'i türetmek rahatsız edici bir seçenektir. Dikdörtgen, karenin özel bir durumu değildir. Tam tersine her karenin bir dikdörtgen olduğu nesnel bir gerçektir ve insan aklına uygun olan kalıtım *Dikdörtgen*'den *Kare*'ye doğru olmalıdır.



Şekil 1: Anlamsal ve yapısal kalıtım.

Şekil 1'de kalıtım ilişkisinin anlamsal ve yapısal yorumunun neden olduğu ilginç durum görülmektedir. Anlamsal ilişkiye analiz safhasında, yapısal ilişkiye ise tasarım ve kodlama safhalarında başvurulmaktadır. Şekildeki iki ilişkinin farklı diyagramlarda bulunması (soldakinin analiz, sağdakinin tasarım belgesinde) çelişkiyi önlese de aynı kalıtım sembolünün tamamen farklı iki ilişki yerine kullanılıyor olması, nesneye yönelik yazılım geliştirme sürecinin yapısallığına gölge düşürücü bir unsurdur.

Bu örnekle ilgili ilginç bir nokta da şudur ki; tasarımda *anlamsal kalıtımı* takip etmek de yeterince iyi bir çözüm sunmamaktadır. Gerçekte bu örnekte kodun nasıl bir yapıya sahip olması gerektiğini bilebilmek için sınıfların tanımlandıkları bağlamı ve hangi metotlarının ne amaçla kullanılacağını incelemek gerekmektedir. Örneğin *kenar* ve *kenar2* özellikleri içinde aynı değerin saklanması yoluyla yapılan bir kare uygulamasında, şeklin alanını veya çevresini hesaplayan metotlar yeniden kullanılabilir. Diğer yandan kenar uzunlukları üzerinde değişiklik yapan metotları yeniden tanımlamak gerekecektir. Büyük olasılıkla bir kenar üzerinde yapılan değişikliğin otomatik olarak diğerine de yansıması istenecektir. Sorun şu ki bu yeni oluşturulan sınıfın (*Kare* sınıfının) hala bir *Dikdörtgen* olduğu şüphelidir. Dikdörtgen örneklerinin kenarları birbirinden bağımsız olarak ayarlanabildiği halde, *Kare* sınıfının örneklerinde bu yapılamamaktadır. Kalıtımın bir sonucu olarak ortaya çıkması gereken bir nesnenin diğeri yerine geçebilmesi özelliğinin var olup olmaması sınıfların kullanıldığı bağlam ile yakından ilgilidir ve dolayısıyla bu tür bir kalıtımın geçerliliği garanti edilemez. Yeni bir bağlamda yeniden kullanılan *Kare* sınıfı kendinden beklenen bazı *Dikdörtgen* davranışlarını gösteremeyebilir.

[9] kalıtım ilişkisi için altı farklı kullanım alanı tanımlamıştır. Üst-sınıfın özel bir durumunu alt-sınıfta tanımlamak (özelleşme) bunlardan yalnızca biridir. Kalıtım ilişkisi bunun dışında aşağıdaki biçimleriyle kullanılmaktadır:

- Somutlaştırmak: Üst-sınıfın sadece tanımladığı davranış, alt sınıfta gerçekleştirilmektedir.
- Kurmak: Alt-sınıf üst sınıfın tanımladığı davranışları kullandığı halde onun bir alt-tipi değildir.
- Genişletmek: Alt-sınıf üst-sınıfın tanımladığı davranışları sürdürür ve geliştirir.
- Sınırlamak: Üst-sınıfın bazı davranışları alt-sınıfta engellenir.
- Birleştirmek: Alt-sınıfta birden fazla üst sınıfın davranışı birleştirilir.

Tüm bu durumlar için aynı kalıtım ilişkisinin kullanılması, otomasyon araçlarına gerekenden çok daha az

bilgi verilmesi anlamına gelmektedir.

Alt-sınıf ile üst sınıf arasındaki gerçek ilişkinin tanımlanabilmesi için bakılması gereken iki ayrı alan vardır. Bunlardan birincisi sınıfların yapısal, bir başka deyişle sözdizimsel özellikleridir. Diğeri ise anlamsal özellikleri yani bir sınıf olarak işlevlerini yerine getirme biçimleridir.

Model güdümlü geliştirme, her şeyin bir model olduğu ilkesine dayanmaktadır [4]. Bu ilkeden hareketle sadece analiz ve tasarım diyagramlarının değil, sınıfların, nesnelerin, kodun ve hatta çalışan programların da model olarak görülebileceği bir model perspektifine sahip olunması, bu paradigmanın uzun vadede elde etmesi gereken entelektüel başarı için gereklidir.

Nesneleri yapısal olarak değerlendirecek olursak en basit halleriyle bir dizi özellik ve metottan oluştuklarını görürüz. Sınıflar ise nesneleri üreten şablonlardır. Nesnelerin özellikleri sınıflarda sadece isim ve tip olarak belirlenmiştir. Metotlar ise sınıf ve nesne arasında genellikle değişmeden kalır.

Sınıflar arasında kalıtım gibi bir ilişki bulunduğunda bu sınıfların türetebileceği nesneler arasında ne tür bir ilişki bulunması gerektiği var olan yazılım geliştirme geleneklerinde net bir yanıtı olmayan bir sorudur. Her iki sınıf da birer nesneler kümesini tanımladığına göre bu soruyu iki parçaya ayırabiliriz.

1. Tanımlanan nesneler arasında nasıl bir ilişki vardır?
2. İki küme arasında nasıl bir ilişki vardır?

Yalnızca yapısal açıdan yaklaştığımızda, kalıtım sonrasında türetilen nesnelerin birbirine eşitliği veya denkliğinden söz edemeyiz. Dolayısıyla sınıfların belirttiği kümeler kaçınılmaz olarak ayrık kümelerdir. Yapısal açıdan bir ilişki tanımlamak için bakmamız gereken yer nesnelerin ve sınıfların içidir.

Bir sınıftan diğeri türetilirken yapılan işlem, var olan sınıfın özelliklerine ve metotlarına yenilerini eklemektir. Bir makineye A'dan B'nin kalıtım yaptığını söylediğimizde makinenin edinebileceği tek bilgi A'nın yorumlayabildiği mesajları (özellikleri okumak ve yazmak da dahil olmak üzere) B'nin de en azından fiziksel açıdan yorumlayabildiğidir. Bu yapısal bilgi derleyiciler için yeterlidir ve alt-sınıfın aynı zamanda bir alt-tip oluşturduğu varsayımı üzerinden hareket ederler. Bu varsayım her zaman doğru olmamakla birlikte basit bilgi sistemleri için genellikle doğrudur. Bir sınıfa yeni özellikler eklemek yoluyla alt-tip oluşturma yöntemi incelenirse, sınıfın mantıksal açıdan yaptığı önermenin sınıfın öğeleri (özellikler) ile olan ilişkisi ortaya konmuş olur.

Alt-tip oluşturulduğuna dair önerme:

$$\texttt{dikdörtgen(X)} \rightarrow \texttt{kare(X)} \qquad (1)$$

Bu mantık önermesi her dikdörtgenin bir kare olduğunu söylemektedir ve bu açıkça yanlış bir bilgi olmakla beraber programcının kalıtım tercihi sayesinde derleyiciye verilen bilgi budur. Bu tür bir yargıya varmayı sağlayan çıkarsama ise nesneye-yönelik programlama paradigmasını gerçekleştiren dil tasarımcıları tarafından dolaylı olarak yaratılmıştır ve sınıfların anlamlarını değil içerdikleri özellikleri ve metotları esas almaktadır. Bu bakış açısına göre her bir özellik, sınıfın örnekleri tarafından uyulması gereken bir kısıt tanımlamaktadır. Bu kısıtların tümü ise sınıfın tanımını vermektedir.

$$\texttt{kare(X)} \leftrightarrow \texttt{özellik(X,kenar,tamsayı)} \qquad (2)$$

Önerme 2'de *Kare* sınıfına ait olma durumu yapısal olarak tanımlanmıştır. Sınıfa ait olmak için gerekli ve yeterli şart *kenar* adında ve *tamsayı* tipinde bir özelliğe sahip olmaktır. Bu gereksinim *özellik* adında üç değişkenli bir ilişki ile gösterilmiştir.

$$\texttt{dikdörtgen(X)} \leftrightarrow \texttt{özellik(X,kenar,tamsayı)} \wedge \texttt{özellik(X,kenar2,tamsayı)} \qquad (3)$$

Önerme 3 aynı yöntemle Dikdörtgen sınıfını tanımladığında *kenar* adındaki özelliğin yanında yine tamsayı tipindeki *kenar2* özelliğini de belirtmektedir. İki gereksinim arasında birleşme (mantıksal VE) işlemi vardır çünkü her bir özelliğin aynı nesne üzerinde yeni bir kısıt tanımladığı düşünülmektedir ve kısıtların tümü aynı anda geçerlidir. *özellik(X,kenar,tamsayı)* ilişkisini p harfi ile *özellik(X,kenar2,tamsayı)* önermesini ise q harfi ile gösterecek olursak:

$$\texttt{kare(X)} \leftrightarrow p \qquad (4)$$

$$\texttt{dikdörtgen(X)} \leftrightarrow p \wedge q \qquad (5)$$

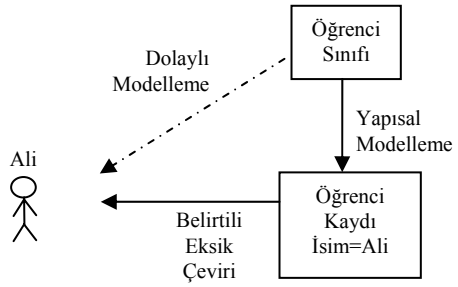
$(p \wedge q) \rightarrow p$ her zaman doğru olduğuna göre 4. ve 5. tanımlar 1 numaralı önermeyi doğrulamaktadır.

Kısacası bir sınıfın özelliklerini o sınıfın örnekleri üzerindeki birer kısıt olarak görürsek, eklediğimiz her özellik ile sınıfın daha özel bir durumunu yaratırız ve bu durumda da kalıtım her zaman bir alt-tip yaratmış olur.

Her dikdörtgenin kare olmadığına dair bilgimiz, sınıfların ve kalıtımın anlamı konusuna farklı bakış açılarının mümkün olduğunu düşündürmektedir. Bu yeni bakış açılarını mümkün kılan, sınıfların ve nesnelerin birer model olarak oynadıkları roldür.

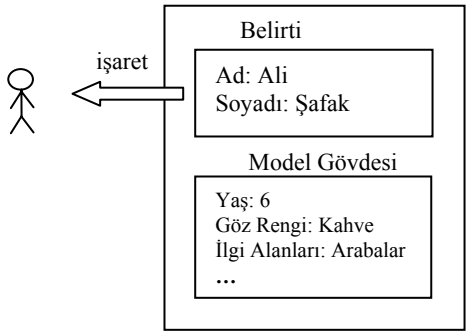
Modelleme insan zihninin çalışma biçiminden kaynaklanan doğal bir yöntemdir ve bir mühendislik terimi olarak anlamı oldukça açıktır. Doğrudan erişmekte zorlandığımız bir sisteme ait bilgilere, bu sistemle benzer bazı özelliklere sahip bir model aracılığı ile ulaşırız. Marvin Minsky [10] modellemeyi şöyle tanımlar: B gözlemcisi için A*'ın A'yı modellemesinin ölçütü, B'nin A hakkında cevaplamak istediği soruları A*'ı kullanarak ne derece cevaplayabildiğidir. Model olmak açıkça bağıl bir durumdur; nesneye yönelik terimle ifade etmek isteyecek olursak; bir roldür. Bu aynı sistemin, iki veya daha çok farklı sistemin modeli olabileceğini gösterir. Bir tasarımda yer alan anlamlı bir sınıf, yerine göre hem geliştirilecek olan kodun, hem de gerçek dünyadaki bir kavramın modeli olarak kullanılmaktadır.

Sınıfları anlamsal açıdan değerlendirmek için programlama dilleri tarafından desteklenen sınıf modelinin içerdiğinden daha fazla bilgiye ihtiyaç vardır. Bir veri tabanı sistemi geliştirilmesi sırasında kullanılan bir *kayıt* sınıfı, sistemin çalışması sırasında yaratılan *kayıt* nesnesinin yapısal bir modelidir. Gözden kaçan nokta ise bu nesne ile bir varlığın kaydının tutulduğu ve dolayısıyla o varlığın da modellendiğidir.



Şekil 2: Varlıkların yazılım tarafından modellenmesi.

Şekil 2'de gerçek dünyaya ait bir varlık olan *Ali* isminde bir öğrenci ile, *Ali*'yi bir bilgi sisteminde temsil eden kayıt nesnesi ve kaydın bir örneği olduğu *Öğrenci* sınıfı arasındaki ilişkiler gösterilmiştir. Sınıf ile nesne arasındaki ilişki *yapısal modelleme* olarak adlandırılmıştır. Sınıf nesne için bir şablon oluşturarak onun fiziksel yapısını belirlemektedir. Klasik kalıtım işlemi bunun gibi bir yapısal modelleme ilişkisinden hareketle yeni bir yapısal modelleme ilişkisi türetmektedir. ne var ki modelleme açısından asıl ulaşılması gereken nokta gerçek dünyadır. Şekilde nesne ile *varlık* (*Ali*) arasındaki ilişki *belirtili eksik çeviri* olarak adlandırılmıştır. *Belirtili* sözcüğü, modelin belli bir birimi modellediğini ve aynı zamanda bu birimin kimliğini gösteren bir işaretçinin modelin içinde yer aldığını ifade etmektedir. *Ali*'nin öğrenci kaydı içinde yer alan bilgilerinden ad, soyadı ve okul numarası gibi kimlik belirten bilgiler modelin gerçek dünyadaki karşılığına işaret etmektedir.



Şekil 3: Belirtili eksik çeviri.

Nesnenin geri kalan özellikleri ise varlığın *eksik çeviri* adını verdiğimiz bir modelini oluşturmaktadır. Eksik çeviri ilişkisi modelleme ilişkilerini sınıflandırmada kullandığımız bir yaklaşımın ürünlerinden biridir. Aşağıda bu yaklaşım anlatılmaktadır.

Modelleme dönüşümü, modellenmek istenen varlığın dönüştürülerek modelin üretilmesi işlemidir. Söz konusu modelleme dönüşümünü bir gerçek varlık ile onun modeli arasında düşünmek yerine, iki ayrı model arasında gerçekleşen bir dönüşüm olarak düşününce, çok daha berrak bir inceleme yapmak mümkün olur. Bir modeli dönüştürerek yeni bir model yaratma sürecinin jenerik bir yöntemi şu şekilde tanımlanabilir:

```
Kaynak modelin her bir özelliği için şunlardan
biri yapılır.
1. Aynen yeni modele kopyalanır.
2. Farklı bir biçimde ifade edilir
3. Görmezden gelinir.
4. Soyutlanır
```

Tüm model için yukarıdaki dönüşüm adımlarından bir veya birkaçını kullanarak dönüşüm gerçekleştirilebilir. Belirli bir dönüşüm türü için hangi tip adımların gerektiği, dönüşümü sınıflandırmak için kullanılabilecek ölçütlerden biridir. Örneğin sadece birinci tipteki adımları kullanarak varlığın tam bir kopyası elde edilir. Birinci ve ikinci tipteki adımları kullanarak ise bir *tam çeviri* yapılır. Görmezden gelme seçeneğini de ekleyecek olursak bir *eksik çeviri* elde ederiz. Alan modellemesinde, gerçek varlıkların ancak eksik çevirisi yapılabilecektir. Bu, orijinal nesne hakkında sınırlı miktarda bilgi içeren bir modeldir.

Çeviri tanım olarak modeli, modellenen varlıkla aynı soyutluk derecesinde yaratır. Bir karakterden, onun ASCII kodunu elde etmek gibi bir işlem *çeviridir*. Eğer modellemenin her hangi bir adımında soyutlama yapılacak olursa bir *kalıp* elde edilir. Bu durumda modellenen elemanın yerine bir değişken ve bu değişkenle ilgili bazı kısıtlar geçer. Sınıfların nesneler karşısındaki durumu kalıp rolünün bilinen bir örneğidir.

Kalıtım ve benzer akıl yürütme teknikleri, model güdümlü bir yazılım geliştirme yaklaşımı çerçevesinde çok daha güvenli ve etkili bir biçimde kullanılabilir. Gerçekte sınıflar ve nesneler de birer modeldir. Fakat modeller tam olarak neyi nasıl modelledikleri konusunda bir bilgi olmaksızın işe yaramazlar. Böyle bir modeli kullanmaya çalışmak, ölçeği bilinmeyen ve üzerindeki renk kodları hakkında herhangi bir açıklama içermeyen bir haritayı kullanmaya benzer.

Model güdümlü yaklaşımda, kalıtım sırasında eklenen özelliklerin oynadıkları roller önem kazanmaktadır. Daha doğrusu ancak ve ancak eklenen özellik, nesne üzerinde sadece ek bazı kısıtların tanımlanmasına olanak sağlıyorsa, yapılan işlem kalıtım olarak adlandırılabilir.

Geometri örneğine yeniden bakacak olursak kare sınıfına ikinci bir kenar bilgisi ekleyerek var olan kare tanımına yeni bir kısıt eklenmemekte, aksine yeni bir boyut eklenmesiyle nesnenin alabileceği değerler katlanarak artmaktadır. Buradaki sorun, eklenen özelliğin *model gövdesine* değil, modelin *belirti* bölümüne ekleniyor olmasından kaynaklanmaktadır. Belirti bölümü bir *işarettir* ve bu örnekte gerçek dünya'daki bir varlığı işaret etmektedir. *İşaret* (sign) kavramı, dilbilim ile yakından ilgili bir bilim dalı olan işaret-bilimi'nin (semiotics) [11,12] temel kavramıdır. Bu bilim dalı sözel, yazılı ve görsel olanlar başta olmak üzere tüm işaretlerle ve işaretlerin anlam oluşturmada kullanımlarıyla ilgilenir. *Belirti* bölümü işaret biliminin kendine özgü kurallarına uyar ve model gövdesi içindeki simetri burada geçerliliğini yitirmiştir.

Belirti ile *model gövdesi* birleştiğinde varlık hakkında tam bir mantıksal önerme oluşturur. Eğer bir kalıp tanımlanmakta ise (örneğin bir sınıf) belirti bölümünün içindeki kalıpların taradığı kümeler kartezyen çarpım işlemine girerek tüm modelleme alanının sınırlarını belirlerler. Nesne gibi bir çeviri modeli söz konusu olduğunda, belirti'ye atanan değerler model gövdesi ile birleştirme (unification) işlemine girerek *belirtili modeli* oluşturur.

```
öğrenci(Ad,Soyadı) →                    (6)
```

```
özellik([Ad,Soyadı],ad,metin) Λ
özellik([Ad,Soyadı],soyadı,metin) Λ
özellik([Ad,Soyadı],yaş,tamsayı) Λ
özellik([Ad,Soyadı],gözrengi,renk) Λ
...
```

Formül (6)'da model-güdümlü bakış açısından bir Öğrenci sınıfının anlamsal içeriği gösterilmiştir. Bu yaklaşımda sınıfın gerçek anlamın kuşatılması hedeflendiğinden, modellenen varlık, bilgisayar belleğindeki sembolik nesne değil gerçek dünyadaki gerçek nesnedir. Mantıksal ifadede parametre olarak basit bir değişken yerine, sınıfın *belirti* bölümünün içeriğini oluşturan sözde *özellikler* (gerçekte nesneyi gösteren bir işaretçinin bileşenleri) yer almaktadır. Bu durumda bir sınıfa kalıtım sırasında eklenen yeni bir özellik olduğunda, bu özelliğin belirti bölümüne mi yoksa model gövdesine mi eklenmesi gerektiğine dikkat edilmesi gerekmektedir. Ayrıca gerçek varlıkların tüm özelliklerine erişmek prensipte mümkün olmadığından, kural tek yönlü gerektirme kullanılarak yazılmaktadır. Bu nedenle nesneye yönelik kalıtımı yorumlarken yaptığımız çıkarsamaların burada yapılması mümkün değildir.

Kalıtım sorununa model güdümlü bir yaklaşım yine kare-dikdörtgen örneği üzerinden tartışılabilir. Model güdümlü yaklaşımda, modellenen varlıklar hakkında varılan yargılar ile modellerin kendisi hakkında varılan yargılar birbirinden tamamen ayrıdır. *Her kare bir dikdörtgendir* önermesinden yola çıkılarak karenin ve dikdörtgenin modellerinin biçimi hakkında bir yargıya varılamaz. Ayrıca bir modelden, tamamen sözdizim düzeyinde tanımlanmış bir dönüşüm yoluyla elde edilmiş bir başka model varsa. Dönüşümün girdisi ve çıktısı olan iki model hakkında anlamsal bir yargıya varmak mümkün değildir.

Tüm bu pratik çıkarsamalar yapılamazken yeniden kullanımın nasıl gerçekleştirileceği akla gelen bir sorudur. Gerçekte model güdümlü yaklaşım potansiyel olarak kalıtım gibi birçok yeniden kullanım sağlayan dönüşümü destekleyebilir. Kalıtım örneğine dönersek, her karenin bir dikdörtgen olduğu bilgisi karelerin de bir dikdörtgen olarak modellenebildiğini gösterir.

Sınıfları birer dil gibi düşünecek olursak, nesneler de bu dilde yazılmış olan ifadeler olarak görülebilir. *Her kare bir dikdörtgendir* önermesi kare dilinde ifade edilebilen nesneler kümesi ile dikdörtgen dilinde ifade edilebilen nesneler kümesi arasındaki bir ilişkiyi (alt-küme ilişkisi) belirtmektedir. Diğer yandan bu ifadelerin kendisi hakkında bir yargıda bulunmak mümkün değildir ve bunu yapmak için her iki dili de bilmek gerekmektedir. Her karenin bir dikdörtgen olduğu bilgisine sahip olan bir geliştirme aracı kare dilinden dikdörtgen diline bir çeviri yöntemi tanımlamak için geliştiriciyi yönlendirebilir. Bu durumda geliştirme aracı, geliştiriciden, girdi olarak *kare* tipinde bir nesne alan ve çıktı olarak *dikdörtgen* tipinde bir nesne üreten bir dönüşüm yazmasını isteyecektir.

Model güdümlü yaklaşım, dil, sözdizim (üst-model), anlam ve dönüşüm gibi kavramlar kullanarak, hiçbir hileye başvurma gereği duymaksızın anlamsal sorunlara doğrudan yaklaşma olanağını vermektedir. Bunun nedeni, nesnelerin tek merkezli yapısına karşılık modellerin tanım itibariyle, *yapı* ve *anlam* olarak iki ayrı varlık biçimlerinin olmasıdır. *Yapı* ve *anlam*'ın birlikte ve ayrı ayrı dönüşebilmesi (ki kalıtım mümkün olan dönüşümlerden yalnızca biridir) sonucu soyut tanımlar daha açık hale gelmektedir.

4. Sonuçlar

Var olan yazılım geliştirme gelenekleri tasarımın ve yazılımın anlamını biçimsel olarak tanımlama olanaklarının kısıtlılığı nedeniyle ihtiyaç duyulan otomasyonu gerçekleştirememektedir. Uzun zamandır süregelen çeşitli sorunlara gereken önem verilmemiş, getirilen çözümler gereken anlamsal derinliği yakalayamamıştır.

Yazılım geliştirme alanında taraftar bulmakta olan yeni bir paradigma: Model güdümlü yaklaşım, sistemlerin ve modellerin anlamı üzerinde çalışma imkanı yaratmaktadır. Sınıfların anlamsal içeriği ile pratik kullanımı arasında doğan çelişkileri çözümlemek için gerekli anlamsal derinlik model güdümlü yazılım geliştirme yaklaşımında mevcuttur.

Bu çalışmada nesneye yönelik paradigmanın önemli kavramlarından biri olan kalıtım konusu kullanılarak yazılım geliştirme sürecindeki anlamsal sığlık vurgulanmış ve bu soruna MGM'nin bakış açısıyla nasıl çözüm getirilebileceği tartışılmıştır. Bu sorunlara yaklaşılırken bazı yeni kavramlar tanıtılmış ve dolayısıyla çalışma aynı zamanda MGM'nin paradigmalaşma hedefine yönelik olarak bir alan çözümlemesi niteliğinde olmuştur. İlerideki çalışmalarla bu fikirlerin proje olarak somutlaştırılması beklenmektedir.

5. Teşekkür

UYMS 2009 organizasyon komitesine gösterdikleri tüm kolaylıklardan ötürü en içten teşekkürlerimizi sunarız.

6. Kaynakça

[1] Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, 2000.

[2] Warmer, J., Kleppe, A.: The Object Constraint Language: Getting Your Models Ready for the MDA. second ed., Addison-Wesley, 2003.

[3] Stevens, P., Pooley, R.: Using UML: Software Engineering with Objects and Components. Addison-Wesley, 1999.

[4] Bezivin, J., On the Unification Power of Models, Software and Systems Modeling, Volume 4, Issue 2, Mayıs 2005, s. 171 - 188.

[5] Sendall, S., Kozaczynski, W., Model Transformation: the Heart and Soul of Model-Driven Software Development, IEEE Software, Special Issue on Model Driven Software Development, pp. 42, Sept/Oct 2003.

[6] Selic, B., The Pragmatics of Model-Driven Development, IEEE Software, Volume 20 Issue 5, 2003, s. 19-21.

[7] C. Atkinson, T. Kühne, "The Role of Metamodeling in MDA" Proceedings of the Workshop in Software Model Engineering (WISME@UML'2002), 2002.

[8] Egesoy, A., Topaloğlu, Y., Yazılım Geliştirmede Model Sınıflandırma Kriterleri, UYMS 2007 Bildiri kitabı, Bilkent Ü. Ankara 2007.

[9] Budd, T., Understanding Object-Oriented Programming with Java, Addison Wesley, California, 1998.

[10] Minsky, M.L., Matter, Mind and Models. Semantic Information Processing, ed Marvin Minsky, MIT Press, 1968.

[11] Ryder, M., Semiotics: Language and Culture. Encyclopedia of Science, Technology and Ethics. ed Carl Mitcham, Macmillan Reference, Temmuz 2005.

[12] Chandler, D., Semiotics: The Basics, Routledge 11 New Fetter Lane, London ,2002.