**EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ**

**(DOKTORA TEZİ)**

# MODEL TABANLI YAZILIM GELİŞTİRME İÇİN SEMİYOTİK BİR MODEL DÖNÜŞÜM DİLİ TASARIMI VE GERÇEKLEŞTİRİMİ

**Ahmet EGESOY**

**Tez Danışmanı : Prof. Dr. N. Yasemin Topaloğlu**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Bilim Dalı Kodu : 619.01.00**
**Sunuş Tarihi : 01.09.2010**

**Bornova-İZMİR**
**2010**

Ahmet Egesoy tarafından doktora tezi olarak sunulan "Model Tabanlı Yazılım Geliştirme için Semiyotik bir Model Dönüşüm Dili Tasarımı ve Gerçekleştirimi" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 01.09.2010 tarihinde yapılan tez savunma sınavında aday oybirliği ile başarılı bulunmuştur.

| **Jüri Üyeleri:** | | **İmza** |
|---|---|---|
| **Jüri Başkanı** | **:** Prof. Dr. Yasemin Topaloğlu | **.....** |
| **Raportör Üye** | **:** Yrd. Doç. Dr. Rıza Cenk Erdur | **.....** |
| **Üye** | **:** Prof. Dr. Oğuz Dikeneli **...** | **.....** |
| **Üye** | **:** Yrd. Doç. Dr.Adil Alpkoçak | **.....** |
| **Üye** | **:** Yrd. Doç. Dr. Murat Komesli | **.....** |

# ÖZET

## MODEL TABANLI YAZILIM GELİŞTİRME İÇİN SEMİYOTİK BİR MODEL DÖNÜŞÜM DİLİ TASARIMI VE GERÇEKLEŞTİRİMİ

EGESOY, Ahmet

Doktora Tezi, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. N. Yasemin TOPALOĞLU
Eylül 2010, 94 sayfa

Bu tezde yazılım geliştirme alanında Model Güdümlü Mühendislik paradigmasının gerçekleştirilmesi için bir dönüşüm dili tasarımı anlatılmaktadır. Paradigmanın gerçekleştirilmesi önündeki güçlüklerden biri, her şeyin bir model olduğu bir geliştirme ortamını düşünmenin zorluğudur. Diğer bir güçlük ise otomasyon içeren bir geliştirme ortamında modellerin sundukları imkanların ifade edilmesi için anlamsal bir dilin var olmayışıdır.

Bu çalışma, her iki zorluğa da işaret-bilimsel (semiyotik) bir açıdan yaklaşmaktadır. Doğal dil çözümlemesinde başvurulan bu bilim dalı, diller tarafından kullanılan temel soyutlama biçimlerini tanımlamaktadır. Bu ilişkiler aracılığıyla modellerin oynayabildikleri çoklu karmaşık rolleri tanımlamak mümkündür. Bir dil öğesi olarak oynayabilecekleri roller belirlenerek modeller, bu roller üzerindeki değişimin belirlenmesi ile de dönüşümler anlamlandırılabilirler.

**Anahtar sözcükler:** Model Güdümlü Mühendislik, Model Dönüşümü, Semiyotik, BLUE-M

# ABSTRACT

## DESIGN AND DEVELOPMENT OF A SEMIOTICAL MODEL TRANSFORMATION LANGUAGE FOR MODEL DRIVEN SOFTWARE DEVELOPMENT

EGESOY, Ahmet

PhD in Computer Eng.
Supervisor: Prof. Dr. N. Yasemin TOPALOĞLU
September 2010, 94 pages

This thesis is about the design of a transformation language for the realization of the model driven engineering paradigm in the software development domain. One of the obstacles of this realization is the difficulty of imagining a development environment where everything is a model. Another obstacle is the lack of a sematic-aware language that may define the possible uses of models in an automated development environment.

In this work, both of these obstacles have been addressed from a semiotical point of view. Being a field of science that is referred for natural language interpretation, semiotics defines the basic abstraction forms used by languages. By using these relations, it is possible to define the multiple complex roles played by the models. The meaning of a model is defined through the roles that it can play as a language element and the meaning of a transformation can be defined through the modification it performs on these roles.

**Keywords:** Model Driven Engineering, Model Transformation, Semiotics, BLUE-M

## TEŞEKKÜR

Bu çalışma süresince değerli yol göstericiliği için tez danışmanım Prof. Dr. N. Yasemin TOPALOĞLU'ya ve hiçbir desteği esirgemeyen kıymetli asistan arkadaşlarıma teşekkürü borç bilirim.

# İÇİNDEKİLER

Sayfa

# İÇİNDEKİLER (devam)

Sayfa

# İÇİNDEKİLER (devam)

Sayfa

# ŞEKİLLER DİZİNİ

# ŞEKİLLER DİZİNİ (devam)

# ŞEKİLLER DİZİNİ (devam)

# ÇİZELGELER DİZİNİ

Çizelge Sayfa

# SİMGELER VE KISALTMALAR DİZİNİ

| Kısaltmalar | Açıklama |
|---|---|
| BLUE-M | Basic Linguistic Unification Environment for Modeling |
| MOF | Meta Object Facility |
| OMG | Object Management Group |
| UML | Unified Modeling Language |
| PIM | Platform Independent Model |
| PSM | Platform Specific Model |

# 1. GİRİŞ

Yazılım geliştirme süreci, farklı uzmanlık alanlarında çalışan insanların rol aldıkları, farklı alanlarda çeşitli soyutlama derecelerinde modelleme görevleri ile örülü, aşamalı bir süreçtir. Model tabanlı yazılım geliştirme paradigması, modellerin yazılım geliştirme süreci içinde oynadıkları belirleyici rolü vurgulamakta ve bir üst seviyeye çıkarmayı amaçlamaktadır. Model tabanlı bir yazılım geliştirme sürecinde modellerin daha düzenli, biçimsel ve bilgisayar destekli kullanımlarıyla birlikte yazılım geliştirme endüstrisindeki hız, kalite, güvenilirlik ve esnekliğin artması, buna karşılık maliyetlerin düşmesi hedeflenmektedir.

Hedeflerin vaat ediciliğine karşın model tabanlı yazılım geliştirme paradigması henüz gelişimini tamamlamamıştır. Öncelikle her şeyin bir model olduğu (Bezivin, 2005) bir yazılım geliştirme anlayışının benimsenmesinde her hangi bir tartışma olmasa da bu konuda yeterince tavizsiz bir yaklaşım sergilenememiştir. Bir diğer sorun da, modellerin ve dönüşümlerin süreç içinde sahip olabilecekleri işlevlerin belirlenmesi amacıyla kullanılabilecek, yeterince biçimsel ve anlatım gücü yüksek bir aracın yokluğudur.

Var olan model dönüşümü yaklaşımında modeller dikey ve yatay yönlerde dönüştürülmektedir. Dikey dönüşüm, süreç içinde soyutluk derecesini değiştiren bir dönüşüm olarak tanımlanmıştır (Mens et al., 2006). Yatay dönüşüm ise farklı diller veya teknolojik standartlar arasında gerçekleştirilen bir tür çeviridir. Dikey dönüşümde, anlamda bir genişleme (soyutlaşma) veya daralma (somutlaşma) söz konusudur. Yatay dönüşümde ise anlamın değişmeden kalması veya mümkün olduğunca az değişmesi istenmektedir. Var olan çalışmalarda benimsenen dönüşüm problemlerinde, dönüşüm işlemi ender başvurulan bir işlem olarak görülebilir ve dönüşümlere giren çıkan modellerin anlamını bilen bir geliştirici tarafından gerçekleştirildiğinden, modellerin ve dönüşümlerin anlamlandırılması gibi bir sorunla karşılaşılmamıştır. Ayrıca var olan çalışmalar nispeten kolay olan yatay dönüşümlerde yoğunlaşmıştır (Kurtev et al., 2002; Christoph, 2004; Königs et al., 2006) ve farklı teknolojilere dayanan geliştirme araçlarının ve çalıştırma ortamlarının birlikte kullanılabilmesi problemine yönelmiştir.

Var olan yaklaşımdan farklı olarak bu çalışmanın hedefleri aşağıdaki öngörülerden hareketle belirlenmiştir:

1. Model güdümlü yazılım geliştirme teknolojileri, otomasyon yardımıyla daha önce görülmemiş karmaşıklıkta yazılım projelerinin gerçekleştirilmesine olanak sağlayacaktır. Yeni yazılım projeleri sürekli artan donanım kapasitesinin daha yüksek bir oranının kullanılmasını sağlayan yapay zeka projeleri, görüntü ve ses işleme, sanal gerçeklik ve doğal dil işleme gibi alanlarda pek çok proje fizibil hale gelecektir. Tipik bir proje, makine desteği olmadan baş edilemeyecek çoklukta ve karmaşıklıkta modellerden oluşacaktır.

2. Söz konusu modellerin büyük bir çoğunluğu başkaları tarafından geliştirilmiş, yeniden kullanılan modeller olacaktır.

3. Her şeyin model olduğu bir geliştirme ortamında tasarım ile kod arasındaki ayrım belirsizleşecek, kodlar gibi (biçimsel) tasarlamak ve tasarlar gibi (soyut) kodlamak mümkün olabilecektir.

Bu gereksinimler bilimsel açıdan iki temel sorunu ortaya koymaktadır:

1. Her şeyin model olduğu bir geliştirme ortamının oluşturulması ve bu tür bir anlayışın geliştiricilerce de benimsenmesi *(birleştirme sorunu)*.

2. Modellerin ve dönüşümlerin, makinelerin yorumlayabileceği biçimsellikteki bir yöntemle anlamlandırılması *(anlam sorunu)*. Böylelikle süreçlerde farklı yapı ve işlev sahibi çok sayıda modelin uyum içinde kullanılması amacıyla otomasyondan yararlanılması.

Bu çalışmada *her şey modeldir* ilkesinin ötesinde, tüm işlemlerin de dönüşüm olarak görüldüğü bir bakış açısı benimsenmiştir. Model kavramının genelliğinin sağlanması için dilleri oluşturan tüm öğelerin model olarak görülebilmesine ihtiyaç olduğu düşüncesinden hareketle dile ilişkin konulara yönelinmiş ve model rollerinin temel sınıfları olarak semiyotik biliminden (işaret-bilim) kaynaklanan bazı temel kavramlar kullanılmıştır. Doğal dil çözümlemesinde başvurulan bu bilim dalı, diller, işaretler ve anlamları arasındaki ilişkiyi incelemekte ve doğal diller başta olmak üzere tüm dillerin çalışma mekanizmalarını çözmeyi hedeflemektedir. Orijinal ilgi alanı doğal diller olan bu bilim dalının kavramlarını bilgisayar ortamında kullanabilmek için bazı değişiklikler yapmak gerekmiştir. Bilgi işlemi ön planda tutan bu bakış açısının

işaret-biliminin bilişimsel gerçekleştirimi alanında da önemli bir katkı sağladığı düşünülmektedir.

İşaret-bilimi kavramları kullanılarak yapılan genellemeye modellerdeki *anlam sorununa* bir çözüm olarak önerilen *mega-modellerin* temel ilişkilerinin belirlenmesinde de başvurulmuştur. Semiyotik işaret etme ilişkisi, model ile modellenen nesne arasında veya modeller arasındaki ilişkilerin tanımlanmasında temel yapıtaşı olarak işlev görmeye uygundur. Bu temel ilişkiler aracılığıyla modeller arasındaki daha karmaşık ilişkileri tanımlamak mümkündür. Dolayısıyla anlamı bilinen varlıklarla kurduğu ilişkiler üzerinden modellerin, kaynak ve hedef modellerin anlamları üzerinden de dönüşümlerin anlamlandırılması imkanı vardır. Bu amaçla modeller arası ilişkilerin gösterildiği *mega-model* olarak adlandırılan modeller kullanılmaktadır. Bu yolla model ve dönüşümler bir veya daha fazla mega-model kullanılarak anlamlandırılmaktadır.

İkinci bölümde *Model Güdümlü Mühendislik* alanı ve *Model Güdümlü Mimari* yaklaşımı genel olarak incelenmiştir. Üçüncü bölümde var olan teknolojik yaklaşımların tanıtılması amacıyla literatüre geçmiş model dönüşüm dillerinden üç tanesi incelenmiştir. Dördüncü bölümde modellemenin genel kavramları üzerine bir çözümleme yer almaktadır. Beşinci bölümde işaret bilimi bilişsel kavramlar eşliğinde incelenmiştir. Altıncı bölümde modeller arası ilişkiler ve mega-model kavramı anlatılmıştır. Yedinci bölümde projenin tasarımı ve gerçekleştirimi üzerinde durulmuştur. Sekizinci bölüm sonuçları içermektedir.

# 2. MODEL GÜDÜMLÜ MÜHENDİSLİK

Model Güdümlü Mühendislik (MDE: Model Driven Engineering) (Bezivin, 2005) nesneye yönelik programlama paradigması ve ona ilişkin yazılım geliştirme yöntem, araç ve standartlarından sonra gelen akımlar arasında, kapsam ve uygulanabilirlik açısından en umutlandırıcı olandır. Model kavramının ön plana çıkacağı bir paradigma geçişi sonrasında yazılım geliştirme çalışmalarına, nesnelerin bir işbölümü içinde birleştirilmesi anlayışından çok, bazı hazır modeller üzerinde işlem ve dönüşümler gerçekleştirme anlayışının egemen olacağı düşünülmektedir (Sendall and Kozaczynski, 2003).

Yazılım geliştirmede model-tabanlı yaklaşımın felsefi temelleri, birleştirme ilkesi (*unification principle*) adı verilen bir anlayışa dayanmaktadır. Bu görüş, basitçe her şeyin model olduğunu söylemektedir. Bezivin, kapsamlı kuramsal çalışması (Bezivin, 2005)'de nesneye yönelik program geliştirme paradigmasının temel ilkesi olarak "Herşey nesnedir" düşüncesini göstermiş, model tabanlı geliştirmenin temel alınması ile bunun "Herşey modeldir" biçiminde bir prensibe dönüşeceğini öngörmüştür.

Geleneksel yazılım geliştirme süreçleri ve güncel nesneye yönelik yazılım geliştirme süreçleri, alan analizi, mantıksal tasarım, fiziksel tasarım, kodlama, test ve bakım gibi ana safhalardan ve bunlar içinde yer alan pek çok farklı etkinlikten oluşmaktadır. Bilinen anlamıyla bir yazılım geliştirme sürecinde, her hangi bir etkinlikte karşılaşılan bir sorunun çözümü, tamamen kendine özgü yöntem, araç ve yetenekler gerektirir. Örneğin alan analizi kesinlikle kodlamaya benzememektedir; aynı şekilde bir algoritma geliştirme görevi ile kullanıcı ara-yüzü tasarımı arasında herhangi bir benzerlik yoktur. Bu etkinliklerin her birinde farklı araçlar ve hatta belki farklı uzmanlar yer alır. Dolayısıyla yaygın yazılım geliştirme paradigmaları, yazılım geliştirme sürecinin her bir aşamasının kendine özgü gereksinimlerine vurgu yapmaktadır. Oysa model güdümlü geliştirme yaklaşımında tüm aşamalar için ortak olan modelleme gereksinimleri önem kazanmaktadır.

Model-tabanlı geliştirme yaklaşımında modellerin birinci derece yapıtlar olarak, yani nesneye yönelik paradigmadaki nesnelerin rolüne benzer bir rolde algılanmaları gerektiği birçok kaynakta (Bezivin, 2005; Selic, 2003; Seidewitz, 2003) açıkça belirtilmiş olmasına karşın, kuramsal makalelerden pratik çözümlerin önerildiği çalışmalara doğru gidildiğinde, birleştirme ilkesine yeterli

derecede bağlılık gösterilmediği görülmektedir. Model kavramı, nesnelerin, sınıfların ve programlama alanındaki daha birçok kavramın ortak bir genelleştirmesi olarak görülmesi gerektiği halde, çoğunlukla bu kavramları sadece konu edinen şematik bir ifade biçimi olarak düşünülmüştür. Bu durumda nesneler, sınıflar ve metotlar, bizzat modelleme yapan araçlar olarak değil, sadece modellemeye konu olan unsurlar olarak görülmüşlerdir.

Birleştirme ilkesi sorununun çözümü için, nesneye yönelik yazılım geliştirme sürecinde yer alan tüm araç, ürün, yarı-ürün ve dokümanların model tabanlı geliştirme paradigması ışığında yeniden ele alınarak analiz edilmesinin, ve her birinin işlev ve çalışma biçiminin biçimsel (formal) model tabanlı terimlerle yeniden tanımlanmasının gerekli olduğu düşünülmektedir. Böylece birleştirme ilkesinden hareketle, model tabanlı yazılım geliştirme sürecinin her aşamasının birbirine benzer ve uyumlu çalışmalardan oluşması ve dolayısıyla Model Güdümlü Yazılım Geliştirme disiplininin tam ve saf bir geliştirme paradigmasına dönüşmesi gerektiği düşünülmektedir.

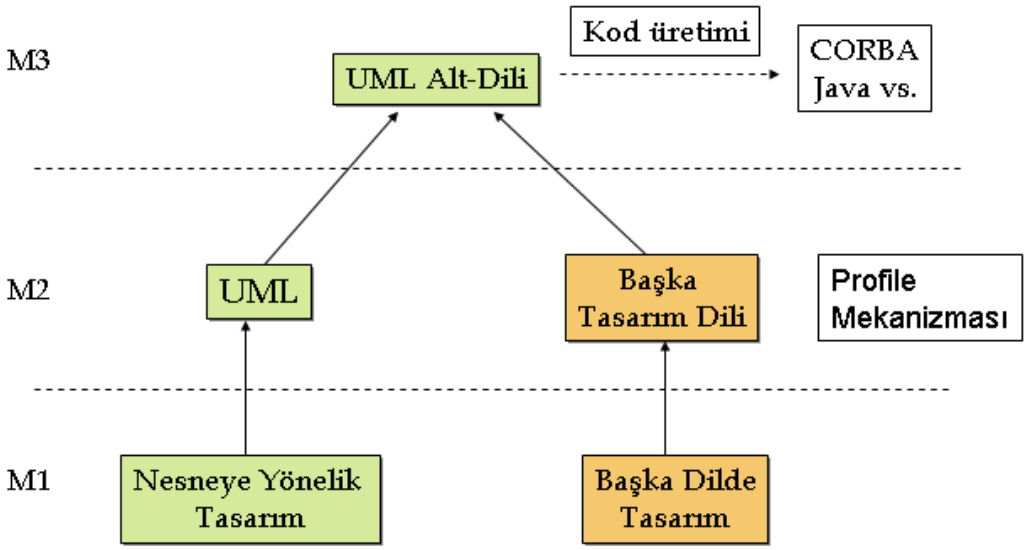
Diğer taraftan bütüncül bir bakış açısı benimsendiğinde tüm bu etkinlikler, geliştiricilerin aktif rol oynadıkları ve içinde modellerin yaratıldığı ve dönüştürüldüğü bir bilgi işlem sisteminin parçaları olmaktadır. Bu sistemin biçimsel (formal) terimlerle açıklanması, ve yazılım geliştirme sürecinin otomatikleştirilebilir saf model tabanlı işlemlere indirgenmesi, model tabanlı yazılım geliştirme paradigmasının uzun vadeli hedeflerindendir (Favre, 2004).

### 2.1 Model Güdümlü Mimari

Nesne Yönetim Grubu (OMG: Object Management Group) 2000 yılının Kasım ayında *Model Güdümlü Mimari* (*MDA: Model Driven Architecture*) projesini (Bezivin, 2005) açıklamıştır. Bu proje, model tabanlı mühendislik dünyasında en çok başvurulan çalışmadır. Proje kapsamında ortaya konulan *Meta Object Facility (MOF)* adı verilen dört soyutlama katmanından oluşan bir üst-modelleme standardı, farklı teknolojiler arasında uyum ve genişletilebilirlik sağlamayı hedeflemektedir. *MOF* Birleşik Modelleme Dili'nin (*Unified Modeling Language: UML*) bir bölümünün özyinelemeli olarak kullanılmasıyla UML ve benzer dillerin tanımlanabileceği düşüncesine dayanmaktadır.

Şekil 2.1'de MOF katmanları şematik olarak gösterilmişlerdir. M1 katmanında tasarımlar bulunmaktadır. M2 katmanında tasarımların yazılmış

olduğu tasarım dilleri ve tabi UML yer almaktadır. Burası meta-model katmanıdır. M3 katmanı ise tüm tasarım dillerinin başvurduğu meta-meta-model düzeyidir. Bu düzeyde UML'in sadece temel öğelerini içeren bir dil tanımlanmaktadır. M2 düzeyindeki tanımlar bu üst dili kullanarak modelleme dilleri tanımlarlar.




Şekil 2.1. MOF katmanları

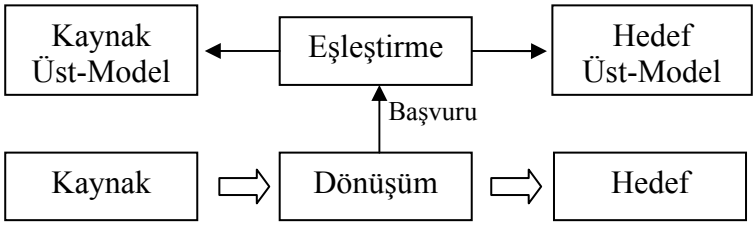
*MOF*'da her bir katman bir altındaki katmanın sözdizimini tanımlar. Bunun istisnası M0 katmanıdır. Şekilde gösterilmemiş olan *M0* katmanında modellenen asıl sistemler bulunur. Buradaki yapılar ile *M1* düzeyindeki tasarımları arasındaki ilişki *benzeşme ilişkisidir*. Diğer katmanlar arasında ise *uygunluk ilişkisi* mevcuttur.

*MOF* ile ilgili bir başka ilginç konu da UML sözdiziminin kendi alt kümesi kullanılarak modellenmiş olmasıdır. Böylece şekilde *UML Alt-Dili* olarak gösterilmiş bulunan, temel UML elemanlarını içeren çekirdek bölümü kendi kendini tanımlamaktadır. Bu yaklaşım estetik açıdan hoş olmakla birlikte, kuramsal açıdan ve pratik yararı bakımından MOF'un eleştiriye açık noktalarından birini oluşturmaktadır.

Alana özgü diller yaratmak istendiğinde, yeni dil için *M2* düzeyinde bir üst-model (meta-model) tanımlanarak *MOF* mimarisine eklenmektedir. Gereken tanımlarda *M3* düzeyine verilen referansların yanı sıra, *M2* düzeyinde bulunan UML tanımlarından ve daha önce tanımlanmış diğer dillerin tanımlarından kalıtım yoluyla yararlanılır. UML'in genişletilmesi için standart bir format sunan *profile*

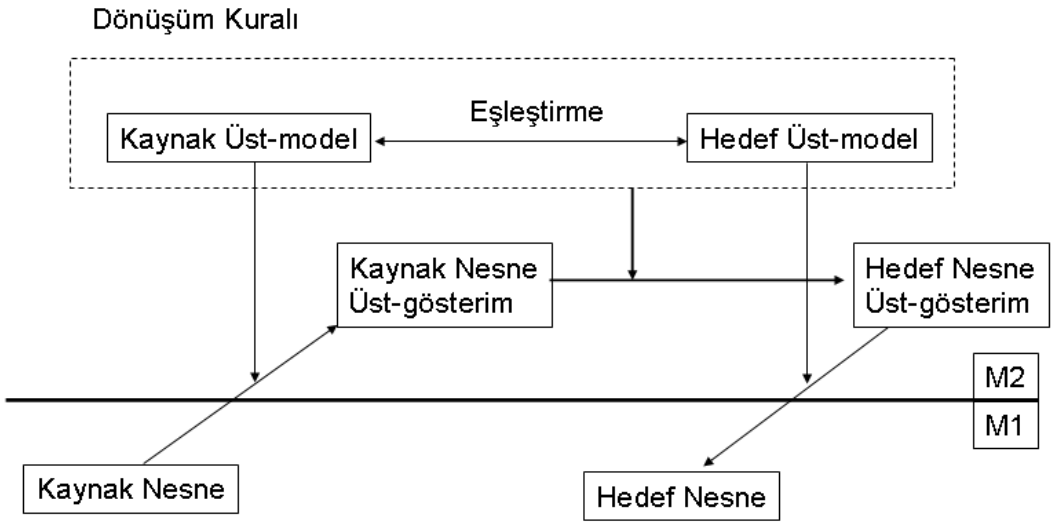
*(profil) mekanizması* çoğu dil tanımı için uygun bir araç sunmaktadır. *Profile mekanizmasının* imkanları kullanılarak çeşitli UML elemanlarının benzerleri (steriyotipleri) üretilebilir ve bunlar üzerinde kısıtlar tanımlanabilir.

MDA tabanlı dönüşüm anlayışında da MOF üst-modelleri önemli rol oynamaktadır. Şekil 2.2'de üst-modeller üzerinden tanımlanmış bir dönüşüm görülmektedir.




Şekil 2.2. MOF tabanlı dönüşüm stratejisi

Şekildeki, üst modelleri eşleştirme üzerinden tanımlanan dönüşüm yapısı, MDA dönüşümlerinin tipik şemasıdır ve açık bir biçimde yatay dönüşümleri ön plana çıkarmaktadır. Ayrıca kaynak ve hedef diller (veya teknolojiler) arasında semantik açıdan tam bir eşdeğerlik gözetilmektedir. Şekil 2.3'te üst-model üzerinden dönüşüm tanımlanması daha detaylı olarak gösterilmiştir.




Şekil 2.3. MOF tabanlı dönüşüm uygulama

Şekildeki örnekte kaynak model önce kendi üst-modeli tarafından okunmakta ve bu okuyuşu yansıtan bir geçici gösterim biçimi oluşturulmaktadır. Asıl dönüşüm eşleştirmeler aracılığı ile bu geçici gösterim biçimi üzerinde meydana gelmekte, ve sonrasında hedef üst-modeli aracılığıyla hedef nesne üretilmektedir.

*MOF* standardı, sözdizimsel tanımlardaki başarısına rağmen modellere ve dönüşümlere anlamsal açıdan yeterli kalite ve tutarlılıkta destek veremediği yolunda yoğun eleştiriler almıştır (Favre, 2004; Alvarez et. al., 2001; Thomas, 2004; Atkinson and Kühne, 2001, 2002). Bunun nedeni büyük ölçüde, insan geliştiriciler için düşünülmüş olan ve ticari kaygılardan arınmış bir gelişim izleme konusunda sıkıntıları bulunan UML diline olan kesin bağımlılıktır.

UML, nesneye yönelik bakış açısını yansıtan bir dildir ve güçlü yanları ile birlikte eski paradigmanın zayıf yanlarını da desteklemektedir:

1. "Nesne" sözcüğünün kelime anlamından anlaşılacağı üzere, nesneye yönelik bakış açısı, adeta modellediği varlığın kimliğini çalan ve kendini onun konumuna yerleştiren bir temel yapıtaşı (nesne kavramı) üzerine bina edilmiştir. Nesneler doğaları itibarı ile birincil varlıklardır. Modellerin ikincil, hatta üçüncül karakteri nesneler için yabancıdır. "Sen orijinal sistemin nesi oluyorsun" gibi metaforik bir sorguya nesnenin vereceği yanıt "Ben oyum" olacaktır. Bu yaklaşım model kavramı ile uyumlu olmayan bir yaklaşımdır. Modeller ikincil rollerinin bir sonucu olarak hem gerçek dünya nesnelerine hem de birbirlerine daha anlamlı referanslar yapma imkanına sahip olurlar ve farklı soyutluk derecelerinde çalışabilirler.

2. Nesneye yönelik yaklaşımda polimorfizm, bir kural değil istisnadır. Sınıfların kaynaştırılması değil ayrıştırılması esastır. Aynı yaklaşımın bir uzantısı olarak *MOF* içinde de modeller ve diller (üst-modeller) birbirlerinden kesin biçimde ayrılmışlardır (Modeller M1'de diller M2'de). Diğer yandan gerçek bir model güdümlü paradigma inşasında polimorfizmin kural, tek biçimliliğin istisna olduğu bir yaklaşıma gereksinim vardır. Bir *X* modeli bir *A* modelinin yapısını tanımlarken, *B* modeli ile benzeşim kuruyor olabilir ve aynı anda *C* modeli tarafından yorumlandığında *D* modelini işaret ediyor olabilir.

3. Nesneye yönelik paradigma, model güdümlü geliştirme paradigmasının esaslarından biri olan “Her şey modeldir” ilkesince dile getirilen saflık hedefinden uzaktır. Nesneye yönelik paradigmada “Her şey modeldir” ilkesi var olmadığı gibi “Her şey nesnedir” ilkesi de ciddi bir uygulama alanı bulmuş değildir. Sınıf modelleri nesne değildir; metotlar nesne değildir; nesneler arasında gidip gelen mesajlar nesne değildir; modelleme dilleri nesne değildir; kod nesne değildir. Nesneye yönelik paradigma yeterince birleştirici biçimde uygulanamamıştır. UML ve dolayısıyla *MOF*’da da birleştirici bir anlayıştan yoksundur.

Bu bölümde hedeflenen nesneye yönelik paradigmanın eleştirisi değildir. Nesneye yönelik paradigmanın bu eksikliklerinin, model güdümlü paradigma için vurgulanmış olan hedefler çerçevesinde ortaya konmuş olduğunu belirtmekte yarar vardır. Nesneye yönelik paradigma ortaya çıktığında henüz model güdümlü paradigmanın var olmadığını ve bu derece büyük otomasyon hedeflerin ortaya konmuş olmadığını yadsımak mümkün değildir. Bu programlama anlayışının yazılım geliştirme sektöründe yakaladığı başarıyı görmezden gelmek de mümkün değildir. Diğer yandan model güdümlü geliştirme paradigmasının getirdiği hedeflere yönelirken eski nesneye yönelik bakış açısının yeterli olacağını iddia etmek farklı bir konudur ve kanımızca isabetli olmayan bir düşüncedir.

# 3. MODEL DÖNÜŞÜMÜ TEKNOLOJİSİ

Var olan model dönüşüm dilleri, çizge gramerlerinin de temelini oluşturan *dönüşüm kuralı* kavramına dayanmaktadırlar. Dönüşüm dillerinin kontrol yapıları, tercih edilen kural anlayışının izin verdiği serbestlik derecesine bağlı olarak, bir ucunda doğrudan çizge gramerlerine dayanan yaklaşımların, diğer ucunda ise dönüşümün akışı üzerinde tam bir kontrole izin veren *programlanmış dönüşüm yaklaşımının* bulunduğu bir yelpaze üzerinde yer alır. Bu tez çalışmasında tercih edilen dönüşüm anlayışı, dönüşüm kavramının *programlanmış dönüşüm* (veya *programlanmış genişletme*) olarak adlandırılan esnek yorumuna dayanmaktadır.

Bu bölümde öncelikle kaynağını çizge gramerlerinden alan dönüşüm kuralı kavramı tanıtılacak ve daha sonra *programlanmış dönüşüm* yorumunun temsilcileri olan PROGRES ve MOLA dönüşüm dilleri ile daha güncel bir dönüşüm dili olan ve farklı bir kural yapısını içeren MOFLON dili incelenecektir.

## 3.1 Dönüşüm Kuralları

Çizgesel gösterim biçimi ve onun türevleri olan şema türleri, yazılım projelerinin tasarımları gibi, semboller ve bağlantılar içeren birçok mühendislik ürününde kullanılmaktadır. Metinlerin sözdizimini tanımlayan gramerler gibi çizgelerin sözdizimini tanımlamak amacıyla çizge gramerleri kullanılmaktadır. Çizge gramerleri, yapıtaşları olan dönüşüm kurallarının yapısına göre aşağıdaki gibi sınıflandırılmaktadır:

- Düğüm genişletme (node replacement) çizge gramerleri
- Çoklu-bağlantı genişletme (hyperedge replacement) çizge gramerleri
- Cebirsel yaklaşımlar
- Programlanmış çizge genişletme sistemleri

Düğüm genişletme yaklaşımı ve çoklu-bağlantı genişletme yaklaşımı, temel gösterim biçimleridir ve bilinen metinsel gramer kuralları ile benzerlik gösterirler. Çizge gramerlerinde çekirdek bir çizge ile başlayıp sürekli işletilen ve her seferinde çizgeyi biraz daha geliştiren bir grup ekleme-çıkartma kuralı vardır. Bu kuralların işletilmesi, biçimsel olarak model dönüşümü kurallarının işletilmesi ile büyük benzerlik taşır.

Bir kuralın ateşlenmesi kuralın sol tarafı ile eşbiçimli bir bölgenin çekirdek çizge üzerinde konumlandırılmasını gerektirir. Bu bölge bir düğümden ibaret ise ortaya çıkan söz dizimi, bir düğüm genişletme söz dizimidir. Aynı şekilde bir düğüm yerine bir bağlantıdan yola çıkarak çizge genişletilmesi uygulandığında çoklu bağlantı (hyperedge) genişletme yaklaşımı izlenmiş olur (Agrawal, 2004). Her iki yaklaşım da kural olarak sol tarafında sadece bir eleman olan kurallar kullanırlar ve bu nedenle bağlamsız (context free) çizge gramerleri olarak adlandırılırlar (Baresi, 2002).

Cebirsel yaklaşımda daha serbest dönüşümler kullanılarak daha yüksek bir anlatım gücüne ulaşılmaktadır. Bu gösterim biçimini bir çizge grameri yaklaşımından çok bir çizge dönüşümü yaklaşımı olarak görmek yaygındır. Bu yaklaşımda çekirdek çizge ile eşleşecek şema (kural sol tarafı), çekirdek çizgeden silinecek olan bölge ve çekirdek çizgeye eklenecek olan düğüm ve bağlantılar ayrı ayrı tanımlanabilmektedir. Bu tanımlama birbirinden ayrı iki çizge ile yapıldığında *double pushout* yöntemi adını alır ve bu durumda eşlenen ana şemadan bazı düğüm ve bağlantıları eksik olan bir şema, silinmesi gereken bölgeleri göstermek için kullanılır. İkinci bir şema da fazladan bazı düğüm ve bağlantılar içerir ve eklenmesi gereken bazı elemanları gösterir.

Dönüşüm kurallarını daha da karmaşıklaştırarak *programlanmış genişletme yaklaşımına* ulaşılır. Böylece çizge grameri hedefinden biraz daha uzaklaşılıp çizge dönüşümü anlayışına biraz daha yaklaşılmış olur. Programlanmış genişletme yaklaşımında kuralların uygulanma sırasını da belirlemek mümkündür. Bunun yanında bir dizi adımı, başarısızlık durumunda yan etkisi olmayan tek bir adım gibi görebilme; ek kısıtlar uygulama ve bir dönüşümün diğerlerini çağırabilmesi gibi özellikler de vardır. Bu yaklaşımın en ünlü temsilcisi PROGRES sistemidir (Schurr 1991).

## 3.2 PROGRES Dönüşüm Dili

PROGRES sözcüğü (İngilizce'deki *ilerleme* anlamına gelen *progress* sözcüğünü andırması yanında) *PROgrammed Graph REwriting System* (Programlanabilir Çizge Yeniden-Yazma Sistemi) isminin kısaltmasıdır. 1990'dan bu yana Almanya'nın Aachen Üniversitesinde geliştirilmekte olan PROGRES 400000 satıra ulaşan büyük bir yazılım sistemidir. Mevcut sürümleri Sun iş istasyonları ve Solaris İşletim sistemi üzerinde çalışmaktadır (Aachen Üniversitesi, 2010).

PROGRES'in işlevlerini tam olarak saptamak amacıyla şu üç görev için kullanılabildiğini not etmek yerinde olur (Aachen Üniversitesi, 2010):

1. Çizge yapısında verilerle çalışan, çok üst düzey programlama dili olarak kullanılmaktadır.

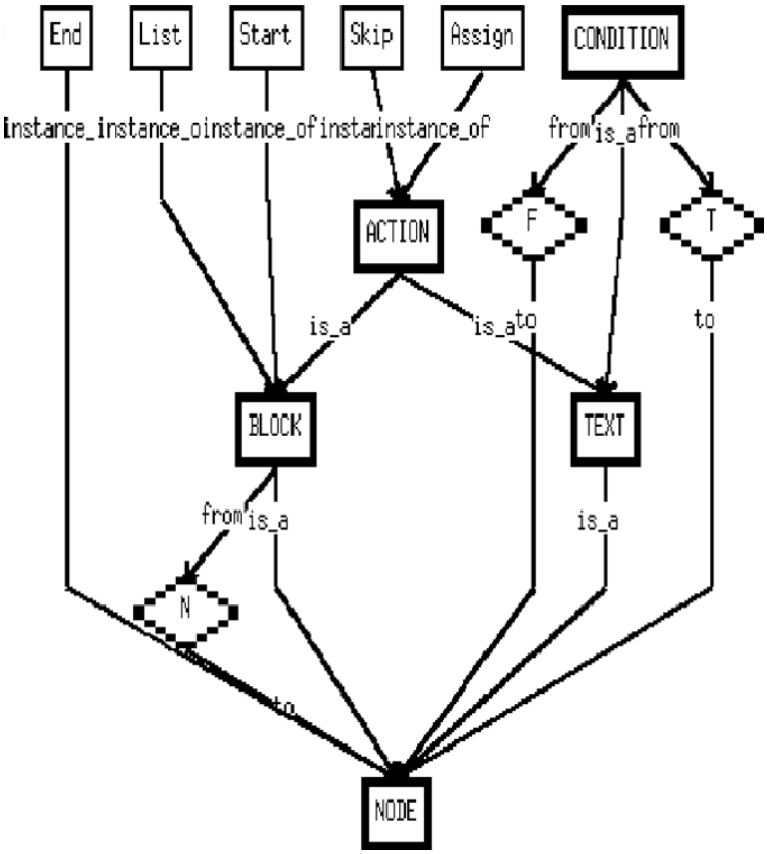
2. Çizge tabanlı veri tabanı sistemi GRAZ için bir görsel veri tabanı programlama dili olarak kullanılmaktadır.

3. Deterministik olmayan dönüşümlerinin hızlı prototiplerini yapmayı sağlayan kural tabanlı bir dil olarak kullanılmaktadır.

PROGRES açıkça bir dizi kuralın rasgele uygulandığı bir platform olmanın yanı sıra, geliştiriciye bir programlama dili ve küçük bir veri tabanı sunan bir platform olarak tanımlanmaktadır ki bu tanım, PROGRES'de dönüşümlerin kodlanması konusunda kullanılabilecek anlatım gücünün yüksekliğini gösterir. İleride göreceğimiz gibi PROGRES'de dönüşümler bir dizi kural ile anlatılmakla kalmazlar. Dönüşüm boyunca gerçekleşen olay akışını prosedürel olarak tanımlamak da mümkündür.

(Schürr, 1994a) 'da bir çizge yeniden yazma sistemi, *terminal ve terminal olmayan sonuçlar arasında ayırım yapmadan bir çizgeler sınıfına ait bir çizge örneğini, aynı sınıfa ait başka bir örneğe dönüştüren kurallar kümesi* olarak tanımlanmaktadır. Çizgelerin aynı sınıfa ait olarak görülmesi çizge grameri bakış açısını işaret etmektedir ve buna paralel olarak da projenin odak noktasını çizgelerin ayrıştırılması konusu oluşturmaktadır.

PROGRES çok paradigmalı bir dil olarak tanımlanmaktadır. Buna karşın şemaların yapıtaşları bilgi gösterim yöntemi açısından nesneye yönelik özellikler taşımaktadır.




Şekil 3.1. PROGRES Şema görünümü (Schürr, 1994a)

Şekil 3.1'de bu yapıtaşlarının tanımlandığı grafik ekran görüntüsü görülmektedir. Örnekte bir *kontrol akış şeması* (*flowchart* eşdeğeri bir şema) için kullanılabilecek gramer tanımlanmıştır. Şemanın üstünde yatay olarak sıralanmış olan kutucuklar şema içinde kullanılabilecek düğümleri göstermektedir (*End, List, Start, Skip, Assign, Condition*). Bunlar *Action, Block* ve *Text* adındaki üç düğüm sınıfını örneklerler veya genişletirler. Sıradüzenin (hiyerarşinin) tepesinde ise *Node* (düğüm) ismindeki sınıf bulunur (şekilde en altta). Şekildeki paralelkenar gösterimler ise şemada izin verilen bağlantı tiplerini göstermektedir. Bağlantı tipleri çıktıkları ve girdikleri düğüm tipleriyle ve bunlara ilişkin çokluk (multiplicity) sınırlarıyla tanımlanırlar.

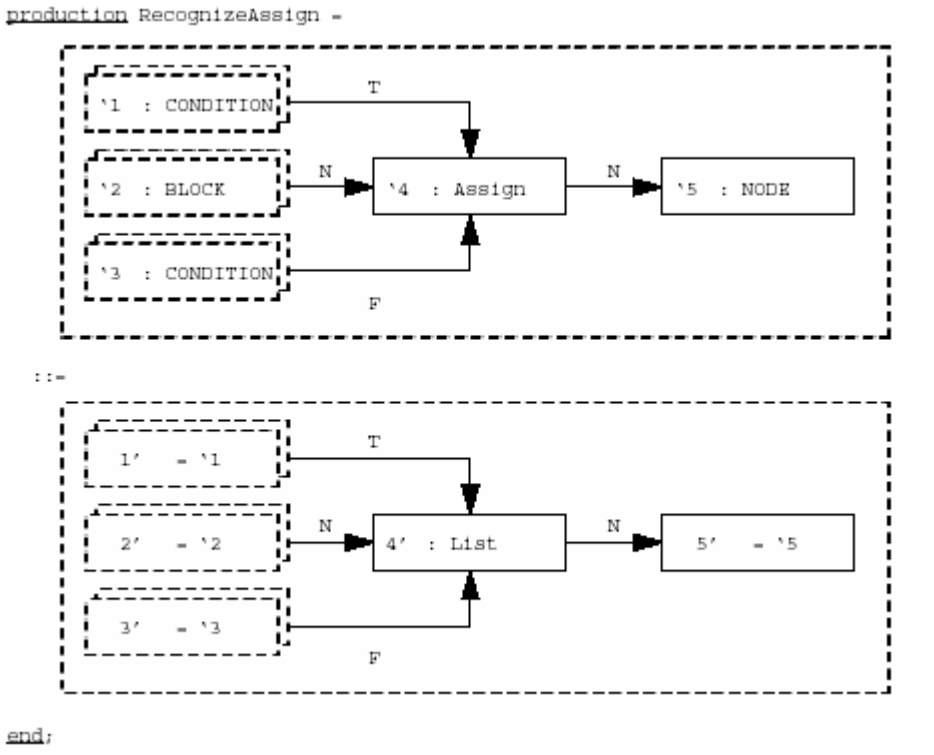
PROGRES arayüzünde şema görüntüsü ile birlikte şemanın metinsel tanımı da yer almaktadır. Metinsel tanım, şematik tanımın anlamsal açıdan eşdeğeridir. Düğüm sınıflarının, düğüm tiplerinin ve bağlantı tiplerinin PROGRES sözdizimi ile tanımları burada görülmektedir. Örnekteki şemanın metinsel tanımı aşağıda verilmiştir:

```
node class NODE end;
```

```
node class BLOCK is a NODE end;
edge type N : BLOCK [0:n] -> NODE [1:1];
node class TEXT is a NODE
  intrinsic
     Text : string;
end;
node class CONDITION is a TEXT end;
node class ACTION is a TEXT, BLOCK end;
edge type T : CONDITION [0:n] -> NODE [1:1];
edge type F : CONDITION [0:n] -> NODE [1:1];
node type Start : BLOCK end;
node type End : NODE end;
node type List: BLOCK end;
node type Assign : ACTION end;
node type Condition : CONDITION end;
```

Örnekte sınıf tanımları *node class* deyimi ile belirtilmiştir. Bunlar arasındaki kalıtım ilişkileri *is a* deyimi ile gösterilmiştir. Kenar (ilişki) tipi tanımları *edge type* deyimi ile düğüm tipi tanımları ise *node type* deyimi ile verilmektedir. Örnekteki *intrinsic* bölümü *Text* düğüm sınıfı için içsel bir veri hanesi tanımlamak için kullanılmıştır. Burada bu düğümün metinsel içeriği saklanmaktadır.

PROGRES'de şema elemanlarının tanımlanmasında nesneye yönelik yaklaşımın oldukça rahat ve sorunsuz bir yorumu benimsenmiştir. Çoklu kalıtım tanımlardaki tekrarları önlemek amacıyla kullanılmaktadır ve daha da önemlisi düğümlerin özellikleri ile bağlantılar birbirlerinden ayrılmışlardır. Böylece nesneye yönelik programlama alanında birçok karışıklığa neden olan bir anlayış terk edilmiştir.

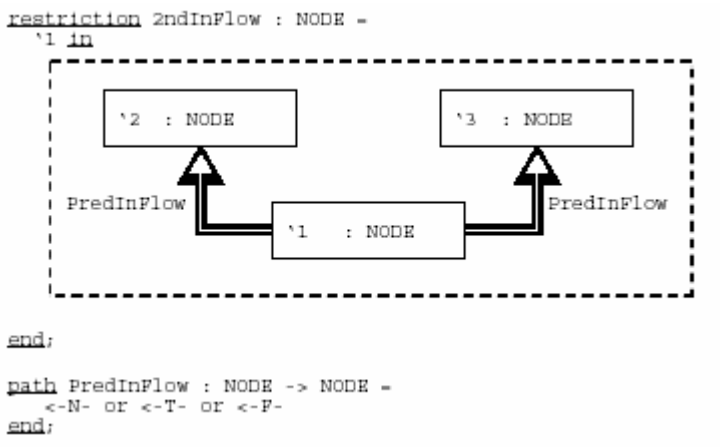



Şekil 3.2. PROGRES dönüşüm kuralı (Schürr, 1994a).

Şekil 3.2'de PROGRES'in bir (atomik) dönüşüm kuralı örneklenmiştir. Şekilde görüldüğü gibi dönüşümlere isim verilir ve önceki durum ve sonraki durumu gösteren bir şema çiftiyle dönüşüm tanımlanır. Şemalar arasında tekrar etmeyen elemanlar dönüşüm sırasında silinir. İki şema arasındaki bağlantılar ortak bir bağlam içinde kullanılan isimler yardımıyla olur. Şekilde görülen *'1, '2* şeklindeki sayılar gerçekte düğümlere takılmış olan isimlerdir. Düğümlerin kesikli ve ikili çizilmiş olması sıfır veya daha çok sayıda eleman içeren kümelere karşılık geldiklerini göstermektedir. Kural yapısı yeniden üretme esasına dayanmaktadır ve değişmeden kalacak olan düğümler için dahi *1'='1* gibi önceki duruma eşitleme tanımları yapılmalıdır.

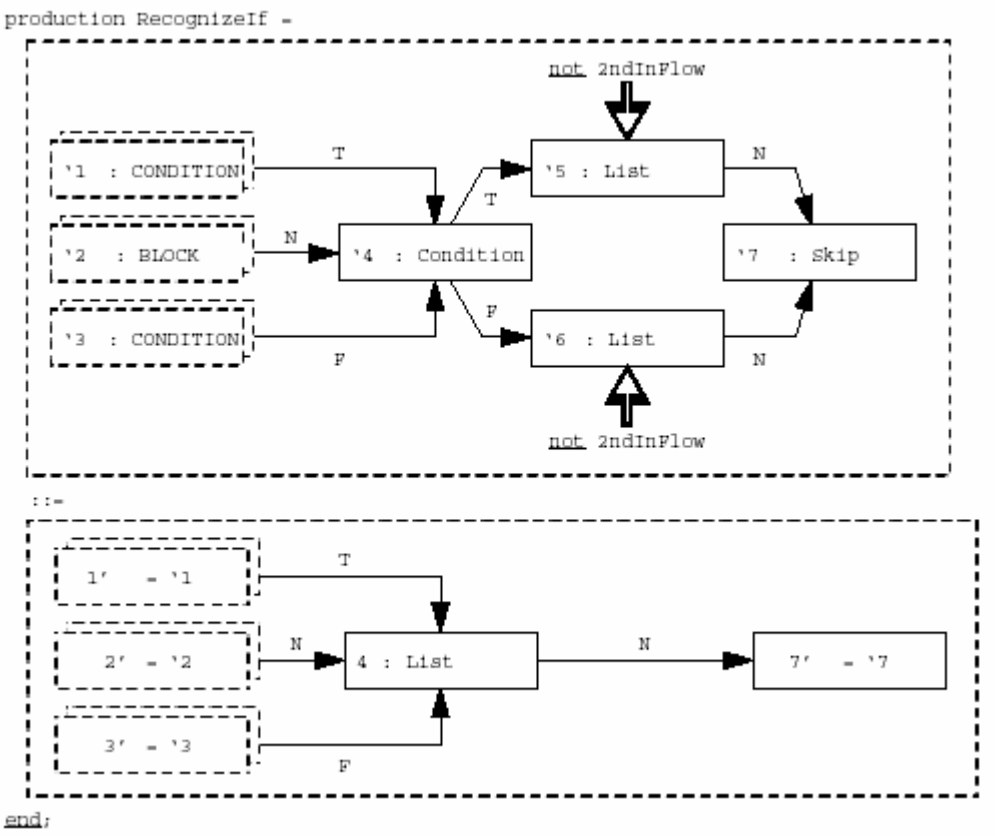
Örnekteki dönüşüm sadece kurallara uygun olarak yazılmış bir *Assign* düğümünü bir *List* düğümü ile değiştirmektedir. Bu kural benzer başka kurallarla birlikte kontrol akış şemalarını ayrıştırarak şemaların düzgün yazılmalarını kontrol etmektedir. Düzgünlük ölçütü *goto* kullanılmadan kodlanabilir olmaya dayanmaktadır. Ayrıştırıcı prensip olarak bir yandan şemanın düzgün yazılmış bölümlerini *List* düğümü ile değiştirmektedir; diğer yandan da kurallarda düzgün yazılmış olması var sayılan bölümler *List* düğümü olarak yakalanmaktadır. Özel bir kural da yan yana yer alan iki *List* düğümünden birini silmektedir. Bu kural kümesi yeterince çok uygulandığında tüm bir düzgün şema tek bir *List* düğümüne

indirgenmektedir. Bu duruma ulaşıldığında şemanın düzgün olduğu anlaşılmaktadır.




Şekil 3.3. PROGRES Kısıt tanımı (Schürr, 1994a)

PROGRES'te belirli özelliklere sahip düğümleri ayırmak amacıyla kısıtlar (restriction) tanımlanabilir. Şekil 3.3'de böyle bir tanım görülmektedir. PROGRES'e ait ayrı bir yapı olan yol (path) kavramı da bu tanımda kullanılmıştır. Tanımlanan kısıtlanmış düğümü temsil eden şema elemanının hangisi olduğu, *NODE='1 in* deyimi ile belirtilmiştir. Anlatılmak istenen bu düğümün en az iki tane giren *N, T* veya *F* tipi bağlantıya sahip olması gerektiğidir. Aslında bu akış şemalarında kaçınılan bir şeydir ve gramerin bütünü içinde negatif olarak kullanılacaktır. Tanımlanan düğüm ile herhangi iki ayrı düğüm arasında *PredInFlow* yolu vardır. Şeklin altında ise bu yol türü bir düğümden diğerine zıt yönde üç tip bağlantı alternatifi olarak tanımlanmıştır. Kısıt tanımına *2ndInFlow* ismi verilmiştir.




Şekil 3.4. PROGRES kısıt tanımının kullanımı (Schürr, 1994a)

Şekil 3.4'de düzgün yazılmış *if-then-else* deyimlerini yakalayan bir başka üretim tanımlanmıştır. Daha önce tanımlanmış olan *2ndInFlow* kısıtı burada kullanılmıştır. *if-then-else* için düzgün yazılma şartı, hem *then* bloğunun hem de *else* bloğunun sadece koşul (*condition*) düğümünden gelen tek bir girişinin olmasıdır. Bu düğümlerin ikinci bir girişlerinin olmadığı *not 2ndInFlow* olarak belirtilmiştir.

PROGRES'de kural çalıştırılmasını kontrol eden deterministik yapılar da vardır. Aşağıdaki *transaction* tanımı bunlardan biridir:

```
transaction RecognizeDiagram
  loop
    RecSkip
    or RecList
    or RecIf
    or RecWhile
  end;
  &RecAxiom
end;
```

Yukarıdaki kodda görünen *transaction* deyimi, kuralların uygulanmasını prosedürel olarak kontrol edebilen bir yapıdır. Örnekteki *transaction*'a *RecognizeDiagram* adı verilmiştir ve daha önce tanımlanmış olan kuralları kullanarak tüm şemanın ayrıştırılması ve sonucun bildirilmesi amacını taşımaktadır. *loop* (döngü) ve *end* (son) deyimleri arasında kalan bölümde bir döngü içinde *or* (veya) deyimleri ile ayrılmış kural çağrıları vardır. Bu biçimde çağırıldıklarında kurallar deterministik olmayan bir biçimde çağırılmaktadırlar. Hangi kuralın şemanın neresine (uyumlu olduğu birden fazla yerden hangisine) uygulanacağı rasgele belirlenir. Diğer yandan PROGRES hangi seçimi yaptığının bir kaydını tutmaktadır ve kural uygulamasının çıkmaz sokağa girdiği durumlarda geri dönerek değiştirebileceği en son kararı değiştirerek yeni bir deneme yapmaktadır. Bu manevra, yapay zekada *backtracking* olarak bilinen yöntemdir ve çözüm uzaylarının derinlemesine (*depth-first*) taranmasını sağlamaktadır.

Mümkün olan tüm kural uygulamaları tükendiğinde döngüden çıkılır. Örnekteki döngü *RecAxiom* adlı bir kuralla birletim (conjunction) ilişkisi içindedir. Döngüden çıkıldığında *RecAxiom*'un doğruluğu kontrol edilir. *RecAxiom* şemanın bir *List* düğümüne indirgenip indirgenmediğinin kontrolünü yapar ve böylece ayrıştırmanın başarısını belirler.

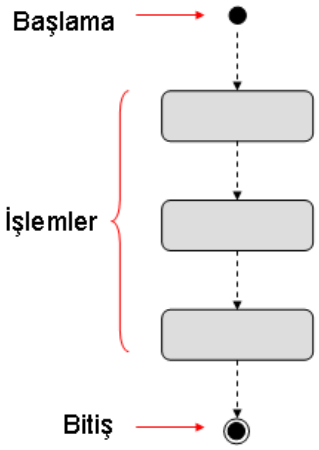
PROGRES'in kural tanımlama dilinin ifade biçimi temelde bildirimsel'dir (declarative). Diğer yandan akış konusunu göz ardı ederek kural yazma olanağının oldukça kısıtlı olduğu gözlemlenmiştir (Prolog'da olduğu gibi). *loop* (döngü) gibi açıkça akış biçimini ilgilendiren yapıların bulunması melez bir yaklaşımı işaret etmektedir.

## 3.3 MOLA Dönüşüm Dili

MOLA dilinin ayırıcı özelliği yapısal programlama, grafik gösterim ve desen tabanlı kuralları birleştirmesidir (Kalnins et. al., 2004). Dönüşümün akışı üzerinde herhangi bir kontrole sahip olmayan bazı yaklaşımların aksine MOLA dili (PROGRES dili gibi) programlanabilir dönüşüm yaklaşımını benimsemiştir ve bu nedenle dönüşümün yazarına (programlayıcısına) geniş bir yetki tanımaktadır.

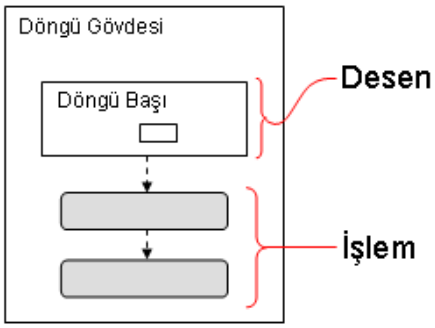
MOLA dönüşüm tanımları, kaynak üst modeli, hedef üst modeli ve Mola şemaları olmak üzere üç parçadan oluşmaktadır. Kaynak ve hedef üst modelleri yürütülen dönüşüm üzerinde bazı kısıtlar uygulamaktadır. Fakat asıl dönüşüm işini yapan bölüm MOLA şemalarıdır. MOLA şemaları uygulama sırasını belirten

oklarla birleştirilmiş görsel komut dizilerinden oluşur. Her bir şema UML (durum şemalarındaki) başlama elemanı ile başlar ve bitiş elemanı ile biter. İkisi arasında bir dizi komut sıralı bir biçimde uygulanır. Şekil 3.5'de bir MOLA şemasının genel yapısı görülmektedir.




Şekil 3.5. MOLA şeması genel yapısı.

*İşlemler*'in her bir örneği, atama gibi basit bir işlem olabildiği gibi, başka bir MOLA şemasına yapılan bir çağrı veya döngü gibi karmaşık bir deyim de olabilmektedir. MOLA'da dönüşümler geliştiricilerin programlama alışkanlıklarına yakın bir biçimde yazılmaktadır. Özellikle döngü yapıları oldukça güçlüdür ve dönüşüm tanımlarının omurgasını oluşturmaktadır.



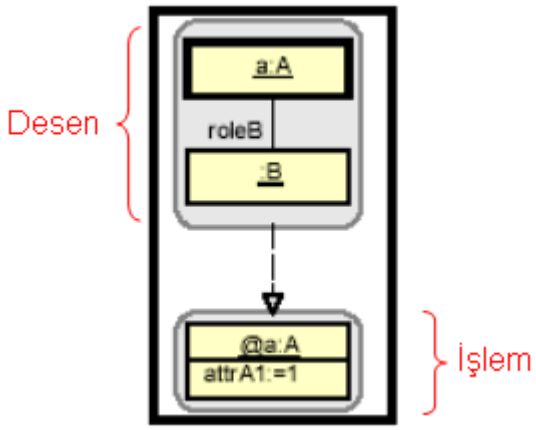

Şekil 3.6. MOLA foreach döngü yapısı.

Şekil 3.6'da Mola'nın döngü yapısının bir gösterimi görülmektedir. Buradaki gibi düz bir kutucuk olarak çizildiğinde, belirtilen *foreach* tipi bir döngüdür. *Foreach* tipi döngü, bir döngü başı ve döngü gövdesinden oluşur. Döngü gövdesi her turda çalıştırılacak olan bir komutlar dizisidir. Döngü başı ise döngünün koşul bölümü gibi işlev görür fakat daha çok kural tanımına

benzemektedir. Döngünün işlev görebilmesi, MOLA'nın örüntü tanıma becerisine dayanmaktadır.

MOLA'da oldukça yalın fakat işlevsel bir örüntü tanıma yaklaşımı tercih edilmiştir. Döngü ifadesinin başında bir desen benzeri yapı vardır ve bu yapının içindeki elemanlardan biri kalın çizgilerle belirginleştirilmiştir. Bu elemana *döngü değişkeni* adı verilmektedir. Döngünün çalışmaya devam etmesi için, bu desen yapısının kaynak model üzerinde, her bir adım için, yeni bir pozisyonda, modelin yeni bir parçası ile eşleştirilmesi gerekmektedir. Eşleştirme işi her zaman kalın çizgilerle işaretlenmiş olan elemanlardan başlayarak yapılır. Bu eleman desen için bir anahtar işlevi görmektedir ve eşleştirmenin herhangi bir bulanıklık içermeden yapılmasını sağlamaktadır.

Şekilde *foreach döngüsünün* bir örneği görülmektedir. *Döngü değişkeni*, *desen* olarak işaret edilmiştir. Döngü içinde yapılan işlem de, altta görülen kutucuk tarafından belirlenmektedir.




Şekil 3.7. MOLA foreach döngüsü örneği.

Şekil 3.7'deki döngü çalıştırıldığında en az bir *B*'ye bağlı olan *A*'ları işaretlemektedir. İşaretleme işlemi *A* tipinden olan ve istenen şartı sağlayan düğümlerin *A1* ismindeki özelliklerine (attribute) 1 değerin atamak biçiminde yapılmaktadır. MOLA'da döngü blokları içinde yerel değişkenler kullanma olanağı vardır. Örnekte küçük harf *a* ismi ile bir değişken (kullanılarak) deklare edilmiştir. Var olan bir değişkene başvurmak için ise isminin başına bir @ işareti koymak gerekmektedir. Örnekte alttaki işlem içinde, yukarıda yakalanan *A* örneğine doğrudan başvurmak amacıyla *a* değişkeni kullanılmıştır.

MOLA'da yer alan bir başka döngü de *while* tipi döngü adı verilen yapıdır. İsim benzerliği dışında programlama dillerindeki *while* döngüsü ile bir ilgisi yoktur. *While* tipi döngü üç boyutlu bir kutucuk (prizma) olarak gösterilir ve yeni yapılar yaratmak yerine var olan yapılar üzerinde değişiklik yapmak üzere tasarlanmıştır. *While* döngüsü *foreach* döngüsüne benzer biçimde çalışır fakat yaratma yerine değişiklik konusuna yoğunlaştığı için, kendisine verilen desenin örnekleri var olduğu sürece dönmeye devam eder. Yapılan değişiklik sonucu verilen desen yok olduğunda döngü sonlanır. Tutarsızlıkların giderilmesi ve bazı karmaşık yapıların silinmesi konusunda yararlı bir özellik olduğu düşünülmektedir.

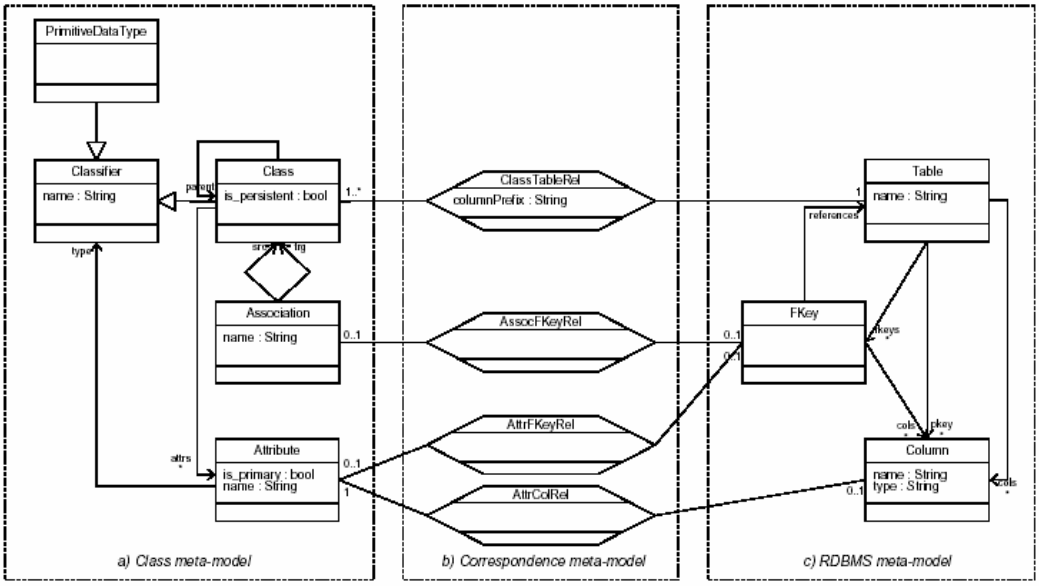
MOLA gerçekleştiriminin veri tabanı temelli olması nedeni ile dönüşümlerin programlanabilirliği veri tabanı sorgu dili SQL'in ifade gücü tarafından sınırlandırılmıştır. Mola gösterim biçimleri sınırlı anlatım güçlerine karşın doğru seçilmiş problem odaklı komutları kullanmaları nedeniyle pek çok önemli dönüşüm tanımını basit bir söz-dizim ile ifade edebilen ekonomik bir dil oluşturmaktadır.

### 3.4 MOFLON Dönüşüm Dili

*MOFLON projesinin hedefi yeni MDA üst-model standartlarına (MOF 2.0, JMI, OCL 2.0 ve QVT ) mümkün olduğu kadar uyan ve bir yandan da güncel çizge dönüşümü teknolojisini yakalayan bir üst-modelleme ve çizge dönüşümü ortamı yaratmaktır* (Moflon Org, 2009). OMG model dönüşümü alanında *QVT* (Query View Transformation) standardını önermiştir. *QVT* standardının bildirimsel (declarative) bölümü Schürr tarafından 1994 yılında ortaya atılmış olan (Schürr, 1994b) üçlü çizge gramerlerine (*TGG: Triple Graph Gammars*) benzemektedir. Formal bir matematik tabanından yoksun olan *QVT* standardını böyle bir temele kavuşturmak üçlü çizge gramerleri yaklaşımının benimsenmesi ile mümkün olmuştur. Üçlü çizge gramerleri, PROGRES'de kullanılan gramer yapısıyla ortak bir kökene sahiptir ve PROGRES takımı içinde yer almış olan bir geliştirici tarafından ortaya atılmıştır. Ne var ki PROGRES içinde kullanılmayan bu yaklaşım, göründüğü üzere MOFLON'un ihtiyaçlarına iyi bir şekilde cevap vermiştir.

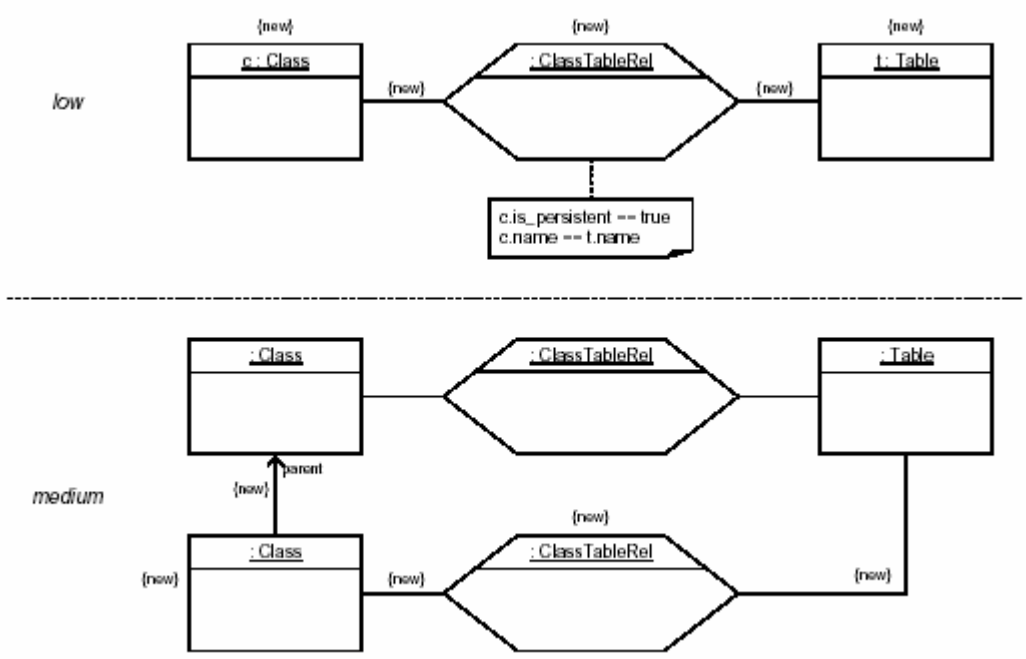
Üçlü çizge gramerleri bir ana üçlü çizge grameri şeması (*Triple Graph Grammar Schema*) ve bir dizi kuraldan oluşur. Üçlü çizge grameri şeması, iki kanadında kaynak ve hedef üst modelleri (ki dönüşümler iki yönlü düşünüldüğü

için her iki kanat da kaynak ve hedef olarak görülebilir) bulunan, ortasında ise bunlar arasındaki karşılık gelme (*correspondence*) ilişkilerinin gösterildiği üç parçalı bir şemadır. Bu şema ile hem dönüşüm görevlerine bir üst bakış sağlanmış olur hem de model üzerinde yapılan değişiklikler sonrasında tutarlılık kontrolü yapmak mümkün olur. Şekil 3.8'de üçlü çizge gramer şeması örneği görülmektedir.




Şekil 3.8. Üçlü çizge grameri şeması (Königs, 2005)

Örnekteki şema, sınıf diyagramları ile veri tabanı arasında bir köprü kurmaktadır. Sol'daki bölüm sınıf diyagramlarının üst modelidir, sağdaki bölüm ise bir veri tabanı üst modelidir. Ortadaki bölüm (altıgen şekillerle gösterilmiş) bağlantı nesnelerini içerir. Bağlantıların üst modellerden ayrı ele alınmış olması anlaşılırlığı arttırmakta ve kod üretiminde kolaylık sağlamaktadır.




Şekil 3.9. Üçlü çizge grameri kuralı örneği (Königs, 2005)

Çizge gramerleri, şemalarının yanında bir dizi de kuralla tanımlanırlar. Şekil 3.9'da üçlü çizge gramerlerinin kurallarının tanımlanması örneklenmiştir. Kurallar birbirine bağlı olan modellerin birlikte değişim geçirme imkanlarını tanımlarlar. Örnekte birinci kural sınıf modeline yeni bir sınıf eklendiğinde veri tabanı modeline de bir tablo eklenmesini söylemektedir. Tablo ile birlikte aradaki bağlantı nesnesi de yaratılarak modele eklenir. İkinci kuralda ise eklenen yeni sınıf var olan bir sınıftan türetilmektedir. Bu durumda sınıf var olan tablo ile ilişkilendirilmektedir; fakat yeni sınıf için yeni bir bağlantı nesnesi yaratılmaktadır.

MOFLON'da kuralların uygulanma sırası statik bir algoritmaya dayanmaktadır. Bu algoritmaya göre öncelikle model elemanlarının bir listesi çıkartılmakta ve daha sonra her bir model elemanı (veya parçaları) üzerinde uygulanabilecek kuralların bir listesi oluşturulmaktadır. Dönüştürülmesi için öncelikle başka adımların atılmasını gerektiren elemanlar listenin sonuna atılmaktadır. Ayrıca aynı eleman üzerinde işlev gören kurallardan öncelik derecesi yüksek olanlar öne konulmaktadır. Elde edilen kurallar model üzerinde uydukları her noktada öncelik sırasına göre uygulanmaktadırlar.

MOFLON dönüşüm dilinde, PROGRES projesinden miras alınan pek çok kavram bulunsa da, dönüşüm anlayışlarındaki temel farklılık dikkat çekmektedir. PROGRES'de dönüşüm kurallarının uygulanmasının genel kontrolünün

prosedürel olarak yapılması bir avantajdır. Bu yaklaşım Moflon'daki kurallara uygulama önceliği verilmesi yaklaşımına göre daha yüksek bir kontrol düzeyi sağlamaktadır. Bir başka önemli fark PROGRES'de tek yönlü dönüşümlerin kullanılması MOFLON üçlü çizge grameri yaklaşımında ise iki gösterim biçimi arasındaki denklik ilişkisine dayanan çift yönlü dönüşümlere yer verilmesidir. Bu yaklaşım, bir araç kullanırken benzer alanlara ve anlatım düzeyine sahip diller arasında geçiş yaparken bir rahatlık olarak görülebilir. Diğer yandan dönüşüm kavramının daha geniş bir yorumu gerektiğinde, örneğin program akışının kendisi girdi ile çıktı arasındaki bir dönüşüm olarak yorumlanırsa, denklik kavramı yeterli olmayacaktır.

Kural yapıları arasındaki dikkat çekici bir fark da PROGRES'deki yapının iki parçalı, MOFLON üçlü çizge grameri yaklaşımının ise üç parçalı şemalara dayanmasıdır. Gerçekte üçlü çizge gramerlerinde ortadaki parça her iki üst-modele de ait olmayıp çift yönlü bağımlılıkları gösteren bir ara yapı niteliğindedir. PROGRES ve onun dayandığı klasik çizge grameri yaklaşımında bu bağlantılar, ortak kullanılan değişken isimleri ile sağlanmaktadır ve dolayısıyla bağlantılar kuralın sağında ve solunda erimiş haldedir. Bağlantıların bu şekilde ayrılmasının bazı yararları vardır. Öncelikle bu yaklaşımda kuralın sağındaki ve solundaki üst-modeller yeniden kullanılabilir yapılar haline gelmektedir. Aynı zamanda dönüşümün kolayca anlaşılan ve izlenen genel bir resmi de ortaya konmuş olmaktadır.

Performans açısından önemli bir nokta da akışı ciddi derecede yavaşlatan özyinelemeli koda (recursion) gerekmedikçe başvurmayarak bunun yerine mümkün olan yerlerde döngü veya kısa yoldan hesaplama yapabilecek fonksiyonlar kullanılmasıdır.

# 4. MODELLEMENİN GENEL KAVRAMLARI

Bezivin (2005) pratik olarak bir modelin ne olduğu sorusunu yanıtlamanın zaman alacağını söylemektedir. Bunun anlamı modelleme kavramını karşılayabilecek genellikte bir bilgi gösterimi gerçekleştiriminin zor olduğudur. Modelleme eski, yaygın, kullanışlı ve geniş kapsamlı bir yöntemdir ve gerçekte bir mühendislik terimi olarak anlamı oldukça açıktır. Marvin Minsky (1968) 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. Modellerin fiziksel yapısı veya model paradigmasının alması gereken şekil hakkında herhangi bir önyargıya sahip olmadan, sadece Minsky'nin tanımını esas alarak ve yazılım geliştirme sürecinin değişik aşamalarındaki bilgi gösterimini ve dönüşümünü analiz ederek, model kullanımını incelemek mümkündür. Bu tür bir inceleme, istenilen genellik düzeyinde ve gerçekleştirim biçiminden bağımsız bir model kavramının giderek daha somutlaşmasını sağlayarak model paradigmasının yapılanmasına katkıda bulunacaktır.

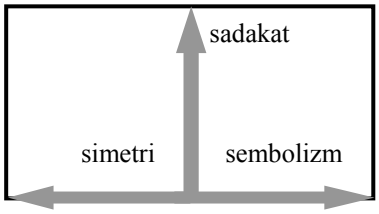
## 4.1 Modellerde Sadakat ve Sembolizm

Modelleme tarzını çeşitlendiren ilk konu, bir sistemin değişik şekillerde unsurlara ayrılabilmesidir. Modelleme sırasında ise her bir unsur için üç seçenekli hareket biçimi mümkündür. Buna göre unsur ya aynen modele yansıtılır, ya kodlanarak modele yansıtılır ya da görmezden gelinir. Görmezden gelme tercihinin bir sonucu olarak modeller, modelledikleri sistem ile ilgili bütün bilgileri barındıramazlar. Bir modelin, anlattığı sistemi hangi başarı derecesinde temsil ettiğinin ölçüsüne modelin sadakati diyebiliriz. Bir modelin sadakatinin ölçülmesi modellenen alana bağımlı bir teknik gerektirse de, elde edilecek değerin sıfır ile bir arasında bir sayı olduğunu ve modellenen sistemden modele yansıtılan bilginin oranını temsil ettiğini öngörebiliriz.

Sistemin modellenen bölümünde ise aynen kopyalama ve kodlayarak aktarma seçenekleri vardır ve bunlardan hangisinin daha çok tercih edildiği de modelin simetri-sembolizm dengesini belirler. Çoğu modelde bazı unsurlar kodlanır, bazı unsurlar ise aynen kopyalanır. Kodlama verinin başka bir söz dizime çevrilerek saklanması anlamına gelmektedir ve bu işlem sırasında herhangi bir bilgi kaybı olmadığı varsayılır. Örnek vermek gerekirse bir coğrafi harita, belli bir bölgenin bir modelidir. Bu modelde yönler, oranlar ve bağıl pozisyonlar aynen kopyalanmışlardır; oysa yükseltiler renklerle kodlanmışlardır, uzaklıklar ise belli

bir ölçekle küçültülerek kodlanmışlardır. Modellerdeki aynen kopyalama pratiği modelin (modellediği sistemle) simetrisini arttırır, kodlama pratiği ise modelin sembolizmini arttırır. Bir modelin içerdiği simetri ve sembolizmi, toplamları bir olan, sıfır ile bir arasındaki sayılarla ifade edebiliriz.

Şekil 4.1'de simetri-sembolizm ve sadakat boyutlarının birbirlerine göre olan konumları gösterilmiştir. Modelde gösterilmesi gereken bilgi miktarının sabit ve sınırlı olduğu düşünülürse, bu bilgi aynen gösterilmezse kodlanarak gösterilmesi gerekeceğinden, simetri ve sembolizm eksenleri zıt yönlü oklar olarak çizilmiştir. kodlanarak gösterilmesi gerekeceğinden, simetri ve sembolizm eksenleri zıt yönlü oklar olarak çizilmiştir.




Şekil4.1. Modellemenin iki boyutu

Bu üst-modelde herhangi bir modelin modelleme performansı, sadakati gösteren 0 ile 1 arasındaki bir sayı ile, sembolizm-simetri dengesini gösteren -1 ile 1 arasındaki bir başka sayı kullanılarak anlatılmaktadır. Dolayısıyla modeller, sınırları belli bir dikdörtgen üzerindeki noktalarla gösterilmektedir.

## 4.2 Nesne-Model Önceliği

Model kavramının geniş tabanlı yorumuna göre modeller birçok değişik rolde karşımıza çıkabilmektedirler. Bu rolleri nesne-model ilişkisine göre kabaca sınıflandırmak mümkündür. Yerine göre nesne veya model birbirleri üzerinde üstünlüğe sahip olabilmektedirler. Modeller öncelikle çeşitli yapıtlar inşa etmek için kullanılmaktadır. Model bu durumda üretilmek istenen nesnenin soyut yapısal bir tanımını verir ve üretim sürecinin sonucu olarak oluşturulan nesne, kendi modelini izleyerek yapılanır. Modelin nesne üzerinde üstünlük kurduğu bu kullanım yazılım dünyasında da çok yaygındır. Üretimsel amaçla kullanılan UML şemaları, tasarım desenleri ve programlama dillerindeki rolleri ile sınıflar bu tür kullanıma örnektir. Bu tür modelleri *izlenen modeller* olarak adlandırıyoruz.

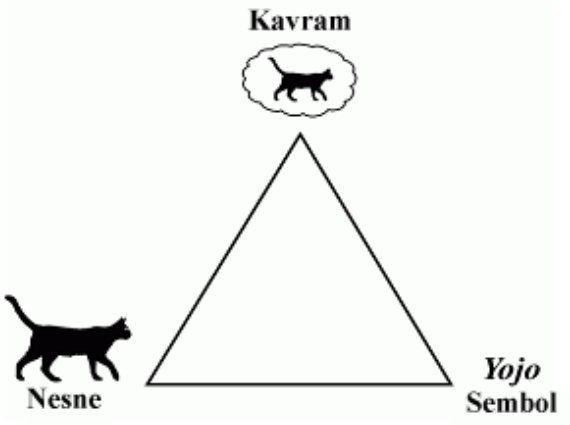
Modellerin bir başka kullanım biçimi, var olan bir sistem veya nesne hakkında bilgi vermek amacını taşır. Bu durumda nesne üzerinde bir

transformasyon gerçekleştirilerek model üretilir ve belirli bir bağlamda model, nesnenin yerine geçerek onun hakkında bilgi verir. Sorgular, değişkenler, alan modelleri ve kullanma kılavuzları bu türden modellerdendir. Bu kullanım biçiminde nesne, modele göre önceliğe sahiptir ve modelin görevi nesneyi izlemek ve yansıtmaktır. Dolayısıyla bu türden modellere *izleyen modeller* adını vermek uygun olacaktır.

Model ve nesnenin birbirleri üzerinde herhangi bir önceliğe sahip olmadıkları üçüncü bir kullanım daha vardır. Bu kullanım tarzında model ve nesnenin her ikisi de aynı anda birbirlerini izlerler ve birbirleri üzerindeki değişikliklere tepki vererek gerekirse kendilerini yeniden düzenlerler. Proxy uygulamaları, referans değişkenleri ve kullanıcı arayüzleri bu türden modeller olarak görülebilirler. Bilgi akışının iki yönlü olduğu durumda ilgili modellere *etkileşimli modeller* adını vermek uygundur. Etkileşimli modeller bileşenlerine ayrılabilirlerse, birbirine bağlı pek çok *izleyen* ve *izlenen* model içerdikleri görülebilir.

# 5. İŞARET BİLİMİNİN BİLİŞSEL ÇÖZÜMLEMESİ

İşaret biliminin en temel kavramı Charles Sanders Peirce tarafından tanıtılmış olan semiyotik üçgen veya Peirce üçgeni olarak tanımlanan yapıdır (Peirce, 1931; Sowa, 2000; Liu, 2000). Semiyotik üçgeninin asıl işlevi bir sembol ile onun sembolizie ettiği varlığı birbirinden ayırmaktır. Bu ikisi üçgenin tabanındaki iki köşeyi oluşturmaktadır. Üçgenin tepe noktasında ise sembol ile nesneyi birbirine bağlayan bir kavram yer almaktadır. Şekil 5.1'de Peirce üçgeninin sık kullanılan bir örneği görülmektedir.



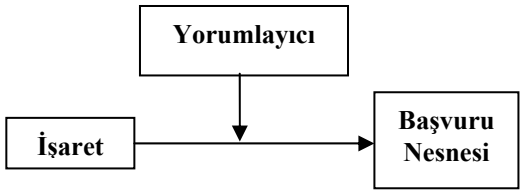

Şekil 5.1. Peirce üçgeni (Sowa, 2000)

Bu örnekte sembolize edilen *Yojo* ismindeki kedidir. Solda kedinin kendisi, sağda sembolize eden isim, tepede ise zihindeki *Yojo* sembolü yer almaktadır. Klasik Peirce üçgeni bilişim alanına pek uygun değildir ve üzerinde değişiklik yapmak gerekmiştir. Dönüşüm kavramını daha iyi destekleyen sembolik bir üçgen kullanmak daha uygun görülmüştür.

Konu Peirce'in üç işaret tipi ile yakından ilişkilidir. *Dizinsel* (indexical) işaretler nesneyi doğrudan fiziksel bağlantı yoluyla gösterirler. *Sembolik* işaretler bazı ön kabuller yoluyla zihinsel bir yorumlama işlemi gerektirirler. *Resimsel* (iconic) işaretler ise işaret ile nesne arasındaki fiziksel benzerlikten yararlanırlar.

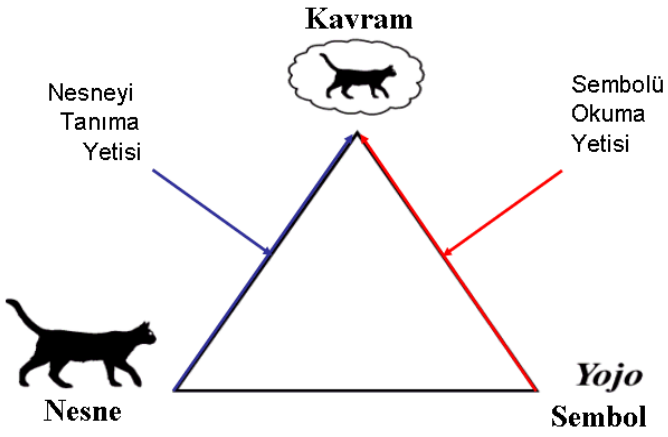
Şekil 5,2'de sembolik işaret tipi gösterilmektedir. Sembolik işaretler yoruma açıktırlar ve bir yorumlayıcıya gereksinim gösterirler. Yorumlama işlevi girdi olarak bir sembol alan ve çıktı olarak bir gösterge üreten (göstergeler dizinsel işaretlerdir) bir fonksiyon tarafından görülür. Bu aynı zamanda temel dönüşüm modelidir. Kısacası sembolik işaret kullanmak istediğimizde ihtiyaç

duyduklarımız, bir sembol ve istendiğinde bunu bir dizinsel işarete dönüştüren bir dönüşümdür.




Şekil 5.2. Sembolik işaret

Gerçekte Peirce üçgeni bölünemez veya indirgenemez bir ilişkiyi anlatmamaktadır. Şekil 5.3'de Peirce üçgeninin sembolik üçgenlere indirgenmesi gösterilmiştir.



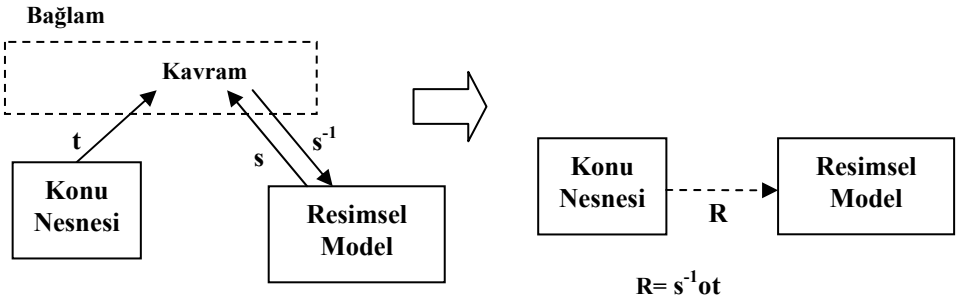

Şekil 5.3. Peirce üçgeninin çözümlenmesi

Buna göre Yojo nesnesi ile Yojo sembolünü birleştiren benzerlik aslında iki adet sembolik ilişkiden oluşmaktadır.

Şekil 5.4'de resimsel işaretlerin çalışma biçimi bir kural biçiminde gösterilmiştir. Resimsel işaretler işaret ile nesne arasındaki benzerliğe dayanırlar. Şeklin sol tarafında resimsel işareti sağlayan benzerlik anlatılmaktadır. Benzerlik kavramına getirilmiş olan işaret-bilimsel yorum aşağıdaki şekildedir:

Benzerlik: İşaretlerin en az bir bağlam içinde eşdeğer olma ilişkisi

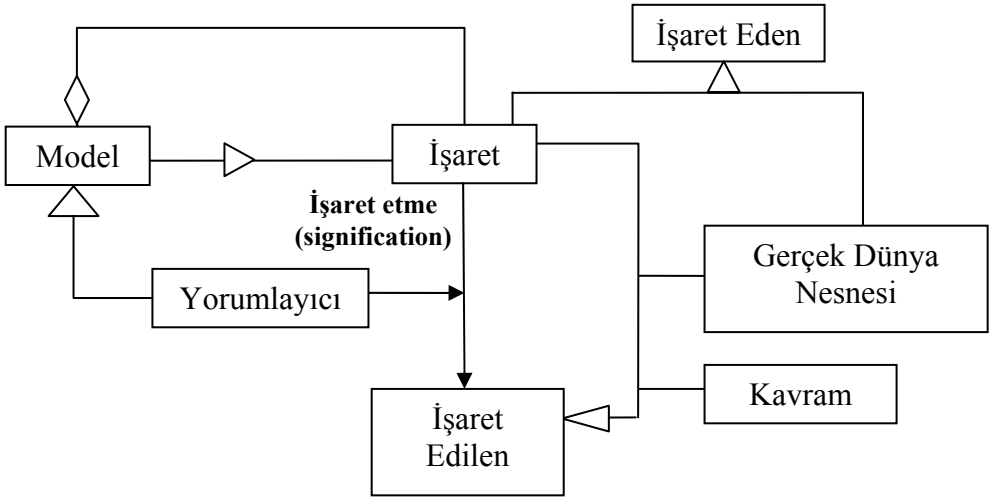
Eşdeğerlik: Verilen bir bağlam içinde aynı referans noktasını (işaret edilen nesne) işaret etme durumu.




Şekil 5.4 Resimsel işaret mekanizması

Dolayısıyla birbirleriyle benzer olan modellerin en az bir bağlamda aynı noktayı (değeri) işaret etmeleri gerekmektedir. Resimsel işaretleri saptayan bir yorumlayıcının bu benzerlik ilişkisinden hareket etmesi gerekmektedir. Yukarıda söz edilen ilişki-fonksiyon çok-biçimliliği sayesinde Şeklin solundaki *Konu Nesnesi* ile kavram arasındaki *t* ilişkisini ve aynı şekilde *Resimsel Model* ile aynı kavram arasındaki *s* ilişkisini fonksiyon (veya dönüşüm) olarak değerlendirmek mümkündür. Buradan yola çıkarak *Konu Nesnesi* ile *Resimsel Model* arasındaki ilişki bileşke fonksiyon olarak yazılır. Bu fonksiyon, örneğin bir sisteme ilişkin belli bir şemanın nasıl üretileceğini betimleyen dönüşüme karşılık gelir. Dönüşümlerde bunun gibi dolaylı yollar izlemek gerekli olmasa da iki yabancı format arasında dönüşüm yapmak istendiğinde sıklıkla izlenen bir yol, kaynak modelin önce, ortak bir standart olarak kabul edilmiş bir dilde yazılması, sonra hedef dile çevrilmesidir. Kaynak ile hedefin söz konusu standart dilde aynı ifadeye karşılık geliyor olması, aralarında bir benzerlik olduğunu göstermektedir.

İşaret türleri ve onların çalışma biçimlerinin, somut tasarımlara yansıtılması çalışmamız açısından önemlidir. Bu amaçla işaretlerin ve işaret edilenlerin kavramsal olarak yer aldığı bir analiz sınıf modeli yaratılmıştır. Şekil 5.5'de, işaretlerin ve ilgili kavramların, çözümlemeye yönelik bir sınıf diyagramı görülmektedir.




Şekil 5.5. İşaret-bilimsel alanın kavramsal analizi

Buna göre işaret edilen sınıfından türetilen işaret, Gerçek Dünya Nesnesi ve de Kavram sınıfları vardır. İşaret eden rolünü ise sadece Gerçek Dünya Nesnesi ve İşaret sınıfları oynayabilmektedir. Ayrıca Model hem kendisi bir işaret'tir hem de işaret'ler içermektedir. İşaret etme ilişkisi İşaret ile İşaret Edilen arasındadır ve Yorumlayıcı bu ilişkiye aracılık etmektedir. Bu bölgedeki okların yönleri aynı zamanda veri akışının yönünü göstermektedir. Alan sadece bilişim sistemlerini kapsadığı için *Yorumlayıcı*'nın da bir model olduğu var sayılmıştır.

Dilbilim ve işaret-bilim gibi ortogonalite ve çokyapılılığın ön plana çıktığı alanlarda analiz yapıldığında kavramsal sınıfların birbirlerine olan bağımlılıklarının yüksek olduğu görülür. Tasarım safhasına geçildiğinde bu sınıfların, tek bir sınıfın oynadığı rollere dönüştürülmesi yaygın bir yaklaşımdır. Çalışmamızda benimsenen yaklaşım da budur.

# 6. MODELLER ARASI İLİŞKİLERİN DÜZENLENMESİ

Projenin *modellerin ve dönüşümlerinin anlamlandırılması* hedefine yönelik olarak, modeller arası ilişkilerin belirlenmesine önem verilmiştir. Klasik anlamdaki nesneye yönelik (UML kullanılarak yazılmış) modellerin aynen sınıflar gibi desen (şablon) işlevine sahip olduğu; yeni model türlerini karşılamak için, dönüşüm, işaret ve mantıksal önerme rolünde modellere de ihtiyaç bulunduğu değerlendirilmiştir. Modellerin birbirlerine ve modelledikleri diğer varlıklara göre sahip oldukları rolleri belirlemek amacıyla *mega-modelleme* yaklaşımından yararlanılmıştır.

## 6.1 Megamodel

Çalışmalarımızda modellerin ve dönüşümlerin projeler içindeki yerlerinin belirlenmesi için kullanılması gereken ilişki tiplerinin belirlenmesi konusunda yararlanılan çalışmalardan en önemlisi Favre ve Nguyen tarafından savunulmuş olan Megamodel kavramı ile ilgili çalışmadır (Favre and Nguyen, 2004). Projemizin amaçları açısından bu çalışma her ne kadar kapsam ve detay yönünden yetersiz olsa da, modeller arası ilişkilere yönelen bir çalışma olması ve bu alandaki terim dağarcığına katkıda bulunması nedeniyle önemlidir.

Yetersizliğin nedeni hedeflerin farklı olmasıdır. Megamodel kavramı Jean Marie Favre tarafından model evrimini ve dolayısıyla model tabanlı yazılım geliştirme görevlerini biçimsel (formal) olarak betimlemek amacıyla ortaya atılmıştır (Favre, 2004; Favre and Nguyen, 2004). Yaygın kullanılan anlamıyla mega-modeller elemanları modeller olan üst düzeyli modellerdir ve modellerin kendi aralarındaki ilişkileri gösterirler. Aslında bu yanlış bir kullanımdır çünkü Megamodel (orijinal yazılışı) deyiminin orijinal kullanımı, bahsi geçen model türünün üst-modelini (meta-modelini) kastetmektedir ve bu bir model türü değil, bu türü tanımlayan belirli bir modelin özel ismidir. Diğer yandan yanlış kullanım literatürde doğrusundan daha yaygındır ve bir çeşit geçerlilik kazanmış durumdadır. Çalışmamızda kelimeyi küçük harfle yazarak ve bitişik yazmak yerine ayraç (*mega-model* biçiminde) kullanarak, model türünün kastedildiği vurgulanmıştır.

Mega kelimesi modellerin tipik olarak üst seviyeli yapılar olduklarını çağırıştırmaktadır. Büyük yapıların aralarındaki ilişkileri gösteren bir model de kaçınılmaz olarak bir mega-yapı olarak görülmüştür. Diğer yandan model-

güdümlü yazılım geliştirme paradigmasının temel doktrini olarak görülen önerme: "*Her şey modeldir*." ilkesi (Bezivin, 2005) modellerin parçalarının da birer model olarak görülmeleri gerektiği düşüncesini doğurmaktadır. Bu durumun bir sonucu olarak Favre'nin mega-modellerinde kullanmayı önerdiği ilişkiler teorik olarak modellerin içinde de işlevsel olmalıdırlar.

Favre'nin yaklaşımı gerçekte pek çok farklı ilişki türü olarak gösterilebilecek durumları beş temel başlık altında toplamıştır. Ortaya atılan χ, δ, μ, τ ve ε ilişki türleri, modellerin birbirlerine göre olan pozisyonlarını göstermek amacıyla kullanıldıklarında, gerçek ilişkiler için bir çeşit tip görevi görmekte ve böylece özünde bir modeller sistemi olan yazılım geliştirme projesinin bir çeşit özetini çıkarmanın mümkün olacağı düşünülmektedir. Bu beş ilişki farklı birleşimlerle (kombinasyonlarla) modeller arasındaki karmaşık ilişkileri tanımlayan bir ilişkisel dil oluşturmaktadır.

Favre ve Nguyen tarafından tanımlanan (Favre and Nguyen, 2004) ilişkiler ve kısa tanımlamaları aşağıda verilmiştir:

δ (delta) : Favre'nin ilk ve en temel ilişkisi δ (delta) sembolüyle gösterdiği parça-bütün ilişkisidir. Bu ilişki sistem ile alt sistemler arasındaki ilişki olarak da görülebilmektedir. Örnek olarak bir otomobil ile o otomobilin direksiyon simidi arasındaki ilişki gösterilebilir. Parça-bütün ilişkisi herhangi bir sistemi oluşturan bileşenleri sıradüzensel (hiyerarşik) olarak düzenler. Büyük parçalar daha küçük parçaların anlamlı birleşimi (kompozisyonu) olarak gösterilir. Parçalardan bütünü inşa etme süreci çeşitli işlemlerden oluşsa da, bu gösterimde yukarıdan aşağıya doğru (kökten yapraklara) giden tüm ilişkiler δ işareti ile gösterilmektedir.

μ (mü) : İkinci temel ilişki modelleme ilişkisidir ve μ (mü) ile gösterilmiştir. μ model ile modellenen nesne arasındaki ilişkidir. Modellemenin yaygın kabul görmüş tanımları model kavramını bir rol olarak görmektedir. Bir nesnenin bir başka nesnenin modeli rolünü oynaması, model görevindeki nesnenin orijinal nesne hakkında bazı bilgileri sağlayabilmesi anlamına gelmektedir (Minsky, 1968). Bu çok gevşek bir tanımdır ve μ ilişkisini işlevsiz hale getirmektedir. Her türlü ilişki, ilişki kuran modeller hakkında bir miktar bilgi içerdiğine göre, kurulan tüm ilişkiler aynı zamanda modelleme ilişkisi olmaktadır. Kastedilen bu olmasa gerektir, dolayısıyla μ ilişkisinin fiziksel bir tanıma kavuşturulması gerekmektedir.

ε (epsilon ) : Üçüncü ilişki küme-eleman ilişkisidir. Küme teorisinden bildiğimiz *elemanıdır* işaretine atıfla ε epsilon işareti ile gösterilmektedir. Dil teorisinin küme teorisine dayandığı ve bu nedenle her dilin bir küme olduğunu ve bu nedenle bu ilişkinin diller ile o dillerde yaratılan ifadeler arasında da geçerli olduğunu savunulmuştur.

χ (çi) : Dördüncü ilişki uygun olma (*conforms to*) ilişkisidir ve χ (çi) ile gösterilmektedir. Bu ilişki modeller ile meta modeller arasındaki durumu betimlemek amacıyla yaratılmıştır. Bir yapısal şablon ve bu şablona uygun olan bir yapıya sahip bir nesne arasındaki ilişkidir.

τ (tau) : Beşinci ve son ilişki, beşli içinde özel bir yere sahip olan τ (tau) ilişkisidir. Bu ilişki bir nesnenin orijinal durumu ile o nesnenin dönüşümden geçirilmiş sürümü arasında bağ kurmaktadır. τ ilişkisi kullanımının son derece esnektir ve mega-modellerin parametresi gibi işlev görmektedir. Model evriminin gerçek akışını anlatan τ ilişkisi anlamsal açıdan fakirdir. Bir dönüşümün kaynağı ile hedefi arasında pek çok değişik ilişki mümkündür. τ ilişkisinin anlamsal açığı diğer ilişkilerle birlikte kullanılması yoluyla kapatılmaktadır.

## 6.2 Modeller Arasındaki İlişkilerin Çözümlenmesi

Favre'nin ilişkileri, model kavramının dar bir yorumunu esas aldığından, anlatım gücünün yetersizliğinden ve ayrıca anlamsal tanımlarının bulanık olmasından dolayı modeller arası ilişkilerin biçimselleştirilmesi için yeterli görülmemiştir. Ayrıca muhtemelen en önemli *mega-ilişki* tiplerine değinilmiş olsa da, bu ilişki kümesine ne tür bir analiz sonucu ulaşıldığı açık değildir.

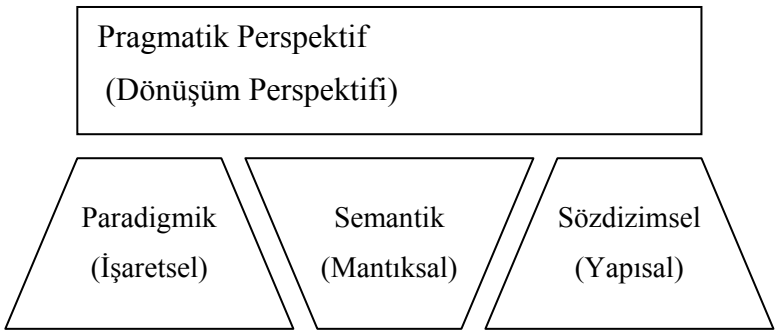
Birleştirme ilkesinde model kavramına yüklenen kapsayıcı anlam, literatürde sıkça örneklenen modellere ek olarak bazı yeni yapıların da birer model olarak incelenmesini gerektirmektedir. Bunların arasında modellerin bölümleri, meta-modeller, kod ve veriler bulunmaktadır. Bu roller ve çevresinde şekillenen ilişkiler, dört işaret-bilimsel işlev perspektifi altında ele alınmıştır. Bunlardan üçü dilsel öğelerle ilgili olan *Paradigmik, Semantik ve Sözdizimsel* perspektiflerdir. Pragmatik perspektif ise daha çok dillerin gerçekleştirimi ve kullanımı ile ilgilidir ve tek başına ikinci katmanı oluşturmaktadır.

Semiyotik perspektiflerin belirlenmesinde, semiyotik biliminin dilleri analiz etmede kullandığı kategorilerden yararlanılmıştır. Her bir kategori, semiyotik

analiz sırasında, analize konu olan metni farklı bir bakış açısı ile ele alarak değerlendirmektedir. Gerçekte bu konuda tam bir söz birliği olmayıp, farklı çalışmalarda farklı kategori grupları önerilmiştir. (Liu, 2000), *fizik, emprik, sözdizimsel, semantik, pragmatik* ve *sosyal* kategorileri önermiştir. (Chandler, 1994)'te ise *modelite, paradigma, syntagmata, retorik varyasyonlar, metinler-arası oluş (intertextuality), kodlama ve sosyal semiyotik* kategorilerini savunulmuştur. Semiyotik analiz alanının içeriği konusunda temelde ortak bir anlayış olmakla birlikte, alanın farklı biçimlerde parçalanması veya genişletilmesi sonucu farklı kategoriler ortaya çıkabilmektedir. Aynı kavram farklı isimlerde de anılabilmektedir.

Semiyotik perspektiflere esin kaynağı olan semiyotik analiz kategorileri, öncelikle bilişim dünyası açısından anlamlı işlevlere sahip olmalıdır. Metinlerin sosyal veya edebi yönünün değerlendirilmesini içeren analiz yöntemlerinin modelleme veya yazılım geliştirme açısından bir önemi yoktur. Örneğin bir Pascal programının sosyal açıdan analizini yapmak anlamsızdır. Bunun yanında bazı kategorilerin diğerlerine dayandırılabileceği veya genelleme yoluyla özdeşlik kurulabileceği görülmektedir. Örneğin bir mesajın fizik yönü ile sözdizimsel yönü arasında bilgi-işlem açısından tam bir kuramsal ayrım yoktur. Bilişim alanına uygunluk ve diğer kategorilerden bağımsızlık açısından değerlendirilerek, bilişim alanının temel semiyotik analiz kategorileri olarak *sintagmatik (syntagmatics), paradigmatik (paradigmatic), semantik (semantics)* ve *pragmatik* kategoriler belirlenmiş ve bunlara karşılık gelen semiyotik perspektifler oluşturularak paradigmanın mimarisine eklenmiştir. *Syntagmata* sözcüğü fazla bilinmediğinden onun yerine *sözdizimsel* terimi kullanılmıştır. Semiyotikte ise sözdizim (syntax) alanı *sintagmatik ve paradigmatik* alanların bileşkesi olarak anılır.

Modelleme rollerinin kaynağının semiyotik biliminde aranması öncelikle şekillendirilmek istenen paradigmanın tamlığı için, doğal dillerin özündeki tamlığın bir örnek oluşturabileceği düşüncesinden kaynaklanmaktadır. Bir başka önemli nokta da, dillerin insanlar için bilgiyi kullanmanın doğal vasıtaları olmaları nedeniyle bir bakıma insanların entelektüel birikiminin arayüzü olarak görülmeleridir. Dil, düşünceyi kapsayan ve ona form kazandıran bir araçtır. Diller veya dillerin parçaları olacak biçimde düzenlenmiş modellerin anlaşılmaları ve kullanımları daha kolay olacaktır. Örneğin tasarım desenlerinin diller oluşturacak biçimde bir arada kullanımları bu alışkanlığı sürdürmektedir.




Şekil 6.1. Semiyotik perspektifler

Şekil 6.1'de dört işaret-bilimsel perspektif gösterilmiştir. *Paradigmik, semantik* ve *sözdizimsel* perspektifler birlikte çalıştıkları için aynı katman üzerinde gösterilmişlerdir. Bu katmanı *linguistik katman* olarak adlandırabiliriz. Mantık önermeleri yapısal olarak, işaretlerin ve desenlerin bir birleşimi olduklarından *semantik perspektif* diğer ikisine göre biraz daha yukarı yaslanmış biçimde gösterilmiştir. Üstte yer alan *pragmatik katman* (perspektif) ise yorumlayıcı rolü çerçevesinde tanımlanan her türlü ilişki ve dönüşümün tanımlandığı yerdir.

Dillerin özyinelemeli yapısı dolayısıyla iki katman tam olarak katmanlı mimarideki (layered architecture) anlamıyla kastedilen gerçek katmanlar olmayıp; aslında aynı anda birbirinin hem üstünde hem altında yer alan iki ayrı faz gibi düşünülmelidir. Bunun nedeni dört rolün her birinin serbestçe istenilen kalınlık derecesindeki (granularity) yapıtlar tarafından oynanabilmesidir. Örnek vermek gerekirse: yol tarif eden bir paragraftaki harflerin her birinin, aynen paragrafın bütünü gibi, birer işaretçi rolü oynaması dolayısıyla *paradigmik* perspektife aittirler. Diğer yandan bu iki kalınlık derecesinin arasında bulunan cümlelerde mantıksal önermelerde bulunuluyor olabilir (*örneğin: "Onun karşısında da bir fırın var!"*) ve dolayısıyla orada da semantik perspektif geçerli olabilir.

### 6.2.1 Sözdizimsel-yapısal perspektif

*Her şey bir yapıdır* (veya *her şey bir sistemdir*) anlayışını yansıtır. Favre'in parça-bütün (δ) ilişkisi etrafında şekillenen ilişkiler bu perspektifte yer alır. Buradaki ilişkiler yorumlamaya ihtiyaç duymayan temel ilişkilerdir. En temel ilişki olarak δ ilişkisi ve ona ilişkin parça ve bütün rolleri hiyerarşik bir biçimde

bir araya gelerek tüm yapıların fiziksel özelliklerini tanımlar. Ayrıca Favre'in meta modelleme ilişkisi χ de bu alana ilişkin temel ilişkilerden bir diğeridir.

Sözdizimsel bakış açısı, parçaların birbirlerine karşı sahip oldukları konumsal ilişkileri temel alır ve semiyotik bilimindeki adı *syntagmatic analysis*'dir. Örneğin bir ifadenin iki biriminin (örneğin kelimesinin) yerlerini değiştirmek bir syntagmatic işlem olmaktadır. (Bazen aynı amaçla syntactic terimi de kullanılmaktadır. Buna dayanarak Türkçe karşılık olarak sözdizimsel terimi seçilmiştir.)

### PARÇ: Parça-bütün ilişkisi

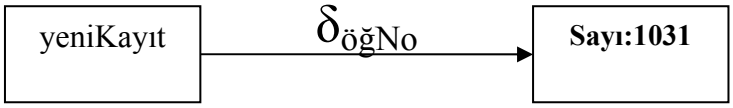
Parça-bütün ilişkisi (δ: delta), yapısal ilişkilerin temelini oluşturur. Sistemlerin yapısı (structure) modellerin ise sözdizimi, daha küçük parçaların birleşmesi ile oluşmaktadır. Bu ilişki bir kayıt yapısı ile kayıtın haneleri arasında ve bir liste ile elemanları arasında mevcuttur.

Yapı belirleyen tanımlar bir hiyerarşi olarak düşünülebilir. Bir sistemi en temel elemanlarına varıncaya kadar alt sistemlerine ayırabiliriz. Elde ettiğimiz hiyerarşi bu sistemin yapısal bir görüntüsüdür. aynı sistemi farklı biçimlerde parçalara ayırarak farklı yapısal görüntüler oluşturabiliriz.

δ ilişkisinin anlamsal içeriğinin istediğimiz düzeye gelmesi için şu özellikleri taşıması gerekmektedir:

- Alt sistemin veya parçanın ismini belirleyebilmek.
- Sistem bir liste ise eleman numarası ile parçayı belirleyebilmek.
- Hiyerarşiyi üreten bakış açısını (çözümleme yöntemini) belirleyebilmek.

Birinci madde örneğin kayıt yapılarında var olan fakat Megamodel'de olmayan bir ifade biçimidir. Çoğu programlama dilinde bir kayıt yapısının bir hanesine *kayıtAdı.haneAdı* biçiminde bir deyim ile başvurabiliriz. Örneğin yeniKayıt.öğNo biçiminde bir ifade yeniKayıt isminde bir değişkenin öğNo ismindeki hanesine başvurmak için kullanılır. Bu özellik orijinal delta ilişkisinde belirtilmemektedir. δ işaretinin altına küçük bir not yazılarak bu eksiklik giderilebilmektedir.




Şekil 6.2. Delta ilişkisi kullanımı

Şekil 6.2'de δ ilişkisinin değiştirilmiş bir sürümü görülmektedir. Bu biçimde yazıldığında sadece parça bütün ilişkisi değil, parçanın bütün içindeki görevi de belirtilebilmektedir. Benzer biçimde isim yerine bir eleman numarası kullanmak amacıyla köşeli parantez kullanılabilir. Üçüncü gereksinim ise klasik programlama dillerinde rastlanmayan bir özelliktir ve en çok nesneye yönelik programlamadaki rol kavramına benzemektedir. Bir bütünü değişik biçimlerde parçalara ayırmak mümkün olduğundan bütün ile parça arasındaki ilişki mutlak bir ilişki değil bütünü parçalara ayıran çözümleme işleminin bir sonucudur. Sistemler gramerler tarafından çözümlenir ve bir sistemi çözümleyen gramer ile sistem arasında çi (χ) ilişkisi bulunur. Basit bir kayıt yapısı üzerinde örnekleyecek olursak grameri bir rol tanımı olarak görebiliriz. öğNo gibi bir öğe söz gelişi yeniKayıt'ın, öğrenci rolüne ilişkin bir parçadır.

ilişki $\delta_{\text{öğrenci/öğNo}}$ biçiminde gösterilmektedir.

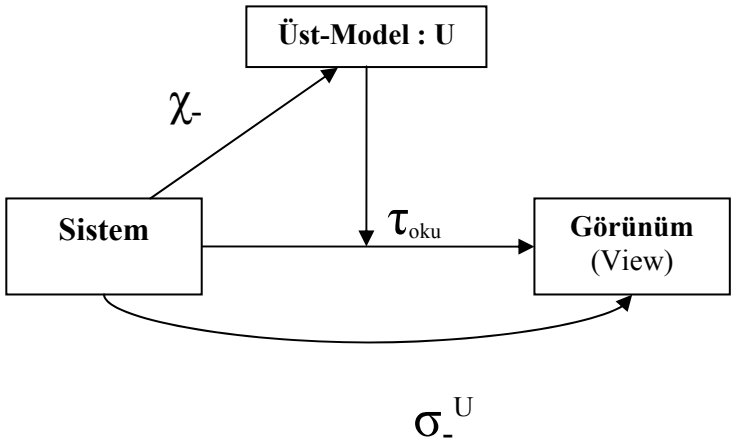
Parça-bütün ilişkisinin kullanıcı tanımlı versiyonları dışında sıklıkla kullanılan hazır biçimleri de vardır. Örneğin parçası olunan yapı dil rolünde olduğunda, dilin desenlerini oluşturan parçalar için *DESENİDİR* ilişkisi, bunun özel bir durumu olan *SIFATIDIR* ilişkisi, dilin sembolleri için *SEMBOLÜDÜR* ilişkisi ve yorumlayıcı rolündeki modelin içindeki mantıksal önermeleri göstermek için *GERÇEĞİDİR* ilişkisi vardır.

### UYAR: Uygunluk ilişkisi χ

Uygunluk ilişkisi χ modeller ile modellerin uyum gösterdiği üst-modeller arasındaki bağlantıyı belirtmek amacıyla ortaya atılmıştır. Ne var ki istenilen düzeyde genelliğe ulaşmak amacıyla daha soyut bir tanım yapmak gerekmektedir.

Bu çalışmada tercih edilen yaklaşımda χ ilişki tipi, bir yapısal şablon ile bu şablona uygun olan yapıdaki bir nesneyi birbirine bağlayan tüm ilişkileri kapsamaktadır. Herhangi bir gramer ile bu gramere uygun olan bir metin parçası arasında da χ ilişkisi vardır. Nesneye yönelik programlamadaki sınıflar ile

bunların örnekleri arasındaki ilişki de χ ilişkisi olarak görülebilir. χ tek yönlü bir ilişkidir ve değişme özelliği yoktur. Gramer rolündeki sistem (üst-model terimini geniş anlamıyla burada kullanabiliriz) bir değişkenler grubu üzerinde kısıtlar tanımlayan bir mantıksal önermeler kümesi olarak görülebilir. Mantıksal önermeler kümesi kullanmak, kısıtlar tanımlamak için temel ve genel geçerliği olan bir yöntemdir. Kullanımı daha yaygın olan gramer, çizge grameri, sınıf ve üst-model gibi bilgi gösterimi yöntemlerinin tümü birer mantıksal önermeler kümesine indirgenebilir.




Şekil 6.3. Okuma dönüşümü

Şekil 6.3'de χ ilişkisi içinde, bir sistem ve onun üst-modeli görülmektedir. Bu çalışmadaki üst-model anlayışı bir sistemin birden fazla üst-modeli olabileceği gerçeğinden etkilenmiştir. Üst model, bir sistemin nasıl çözümlenebileceğinin bir planını vermektedir. Bu plan çerçevesinde parçalar ve onların parçaları tanımlanmakta ve adlandırılmaktadır. Diğer yandan pek çok sistem için çözümlemenin farklı biçimleri vardır ve bir üst-model bir sistemi ancak kendi bakış açısından tanımlamaktadır. Bu tanımların sistemin eksiksiz bir resmini vermesi gerekmemektedir. Özellikle gerçek dünyadaki sistemler hiçbir zaman eksiksiz olarak modellenemezler. Şekil 6.6'daki $\chi_-$ ilişkisindeki '–' işareti böyle eksik bir modellemeyi işaret etmektedir. Eksik bir χ ilişkisi bir sistemin tüm öğelerini kapsamamaktadır.

### 6.2.2 Paradigmik perspektif

Paradigma terimi semiyotik biliminde farklı değerler alabilen sözdizimsel yapıların incelenmesi anlamında kullanılır. (Paradigmic analysis). Değer kavramı ve bir değere sahip olma işlevi temel semiyotik kavram olan işaret kavramı ile

ilgilidir. Bu çerçeve basitçe bir modelin verilen bir yorumlayıcı için (bu yorumlayıcının temsil ettiği alan üzerinde) neyi gösterdiği ile ilgilenmektedir. Dolayısıyla bir modelin bir bağlamda aldığı değer önem kazanmaktadır. Gösterme ilişkisi temel soyutlama birimidir ve değişik desenler biçiminde yapılanarak daha karmaşık ilişkileri oluşturmaktadır. Gösterme ilişkisi kısaca sigma (Σ) ile gösterilir. Örneğin benzerlik ilişkisi μ daha önce anlatıldığı gibi ortak bir hedefi gösteren işaretçiler biçiminde tanımlanmıştır.

### İŞRT: İşaret etme ilişkisi Σ (sigma)

Sigma ilişki tipi basit işaret etme ilişkisi ve ondan türetilen ilişkilerden oluşmaktadır. Kısaltma olarak büyük harf sigma (Σ : signifikasyon'u çağırıştırdığı için) kullanılmaktadır. Σ işaret ile işaret edilen arasındaki ilişkidir. Bunu sembol ile değer arasındaki ilişki veya daha da genelleştirerek sözdizim ile anlam (semantik) arasındaki ilişki olarak düşünebiliriz.

Semiyotik (işaret-bilimsel) ilişkiler Favre'nin mega ilişkileri arasında bulunmamaktadır. Uygun düzeyde soyutlama yapabilmek için bu tip ilişkilere (ve dönüşümlere) ihtiyaç duyulmaktadır. Semiyotik ilişkiler dillerin yapıtaşlarını oluşturan işaretlerin çalışma biçimlerini gösterdiği için kuramsal önem taşımaktadır. Modeller içinde yapılan soyutlamalarda kullanılmanın yanı sıra diğer ilişkilerin çalışma biçiminin anlaşılması için bir yapıtaşı görevi görmektedir.

İşaret etme ilişkisinin bir özelliği ikili değil, bir yorumlayıcının da araya girmesiyle üçlü bir ilişki olarak gösterilmesidir. İşaret etmek en basit anlamda bir soyutlama sağlamaktadır ve bu soyutlama sayesinde işaret eden ile edilen arasında saymaca bir bağlantı kurulmaktadır. Bu bağlantı herhangi bir fiziksel dayanağa sahip olmadığında ancak bir yorumlayıcı ile açıklanabilmektedir. Söz konusu yorumlayıcı işaretin üyesi olduğu dilin yorumlayıcısıdır.

Sigma ilişkisinin en temel uygulaması bellekteki bir adresi gösteren bir işaretçidir (pointer). Bu anlamı en açık olan modelleme öğesidir. Semiyotik biliminde işaretçilerin metaforik bir rolleri bulunmaktadır. İşaretçi (pointer) yerine işaret (sign) terimi kullanılmaktadır fakat bir işaretin anlamı onun tarafından gösterilen bir adres gibi düşünülmektedir. Sembolik işaretlerde gerçekte gösterilen bir bellek adresi yoktur. Bu durumda iken, örneğin gösterilen nokta üzerinde herhangi bir değişiklik yapma olanağı da yoktur. Söz konusu olan tamamen sembolik bir gösterimdir ve iki birimin aynı noktayı gösteriyor olması gibi

durumlar ancak dolaylı yöntemler kullanılarak belirlenebilir. Örnek olarak tüm tamsayı değerlerinin sayı doğrusu üzerinde birer noktayı gösterdiklerini söyleyebiliriz. Sayı doğrusu aritmetikte kullanılan hayali bir kavramdır ve bazı aritmetik ilişki ve işlemleri açıklamak için yararlıdır.

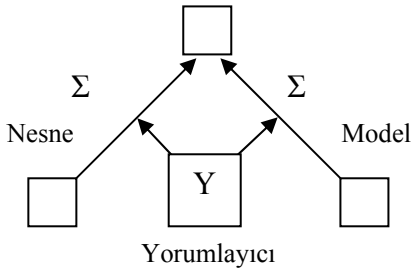
Sigma'nın bir parametresi işaretin oynadığı role ilişkindir. Aynı işaret farklı alanlarda farklı noktaları işaret edebilmektedir. Söz gelişi C programlama dilinde bir rakamsal ifade hem bir sayıyı hem de bir karakteri (hatta bir mantıksal değeri) ifade edebilmektedir. Benzer biçimde bir UML sınıfı hem program içindeki bir Java sınıfını hem de alan (domain) içindeki bir kavramı ifade edebilmektedir. Bu nedenle işaretçilerin anlam bulanıklığı yaratmayan bir kullanım biçimine ihtiyaçları vardır ve bu işaretçi ile bir yorumlayıcıyı eşleştirmek yoluyla yapılır. Bu amaçla sigma'nın üstüne yorumlayıcı adını not ederiz $\Sigma^{alan}$ gibi. Bu ifade işaretçinin *alan* adındaki yorumlayıcı tarafından yorumlanacağını ve bunun sonucunda hedefe ulaşmanın mümkün olacağını söylemektedir. Anlam bulanıklığı tehlikesi olmayan yerlerde yorumlayıcı belirtmeye gereksinim yoktur ve varsayılan (default) yorumlayıcı kabul edilir.

### SİMT: Simetri ilişkisi μ

Model tabanlı yazılım geliştirme alanında karşımıza çıkabilecek orijinal modelleme ilişkisi model ve sistem arasında geçerli olan duruma ilişkin herhangi bir kısıt öne sürmemektedir. Model sistem hakkında bilgi verebilen herhangi başka bir sistemdir (Minsky, 1968). Bu geniş tanım, gözden kaçabilecek bir ilişkiye sahip iki sistemi potansiyel olarak birbirinin modeli haline getirir. Model olmak ilişkisi modelleme dilinin yalın bir öğesi olamayacak kadar geniş bir kavramdır. Bu nedenle μ ilişkisini tanımlarken klasik modelleme ilişkisi yerine, matematikten zaten tanıdığımız simetri ilişkisini ifade etmek daha doğru olacaktır. μ ile ifade edilen klasik modelleme ilişkisi gerçekte model ile modellediği nesne arasındaki simetriyi ifade etmektedir. Bir başka deyişle, model ile nesne arasında bazı ortak noktalar vardır.

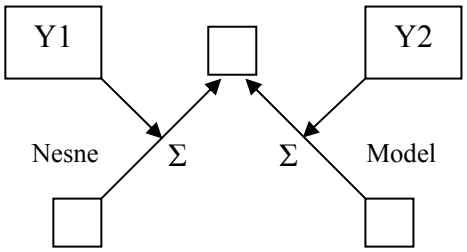
Doğal olarak hem model hem de nesne, gerçekte karmaşık sistemler olduğundan *benzerlik* gibi bir kavramın genel bir tanımını yapmak güçtür. Diğer yandan Σ (işaret etme) ilişkisini kullanarak soyut bir tanım yapmak mümkündür. Simetri ilişkisini göstermek için işaret etme ilişkisi, simetrinin her iki tarafı için aynı hedefi işaret edecek biçimde kullanılır. Σ tabanlı bu tanım en yalın

durumunda iki modelin aynı dilde aynı anlama gelmesini gösterir. Bu ilişki *yapısal simetri* olarak adlandırılmaktadır.




Şekil 6.4. Yapısal simetri ilişkisi

Şekil 6.4'de yapısal simetri durumunun Σ ilişkisi ile tanımlanması gösterilmiştir. Buradaki durum resimsel gösterme ilişkisinin şemasına benzemektedir. İki sistem arasındaki simetri aynı olmasına rağmen amaç farklıdır. Resimsel gösterme ilişkisi Σ'nın bir türüdür ve benzerlik aracılığıyla sadece bir işaretçi oluşturmayı amaçlamaktadır. Burada ise amaç modellemedir. Yapısal simetri deseninin içerdiği benzerlik tek bir ortak yorumlayıcının varlığını gerektirdiğinden oldukça temel bir özdeşlik söz konusudur. Yapısal simetri içeren dönüşümler genellikle anlamı ve kullanılan dili sabit tutarak, sözdizimi kolaylaştırma amacı güderler. Aritmetik ifadeler üzerinde yapılan sadeleştirmeler bu türden dönüşümlerdir. Şekil 6.4'deki yapısal simetri ilişkisi $\mu^Y$ olarak etiketlenmiştir ve Y yorumlayıcısına göre ilişkinin iki tarafının eşdeğer olduğunu göstermektedir.




Şekil 6.5. Anlamsal simetri ilişkisi

Simetri yapısal düzeyde olmasa da anlamsal düzeyde gerçekleşebilir. Bu durumda model ile nesne, fiziksel olarak herhangi bir benzerliğe sahip olmadığı

halde *anlamsal simetriye* sahip olabilir. Anlamsal simetri sadece verilen iki modelin aynı anlama gelmesidir ve modellerin yazıldığı diller farklı olabilir. Şekil 6.5'de anlamsal simetri ilişkisi gösterilmiştir. *Y1* ve *Y2* olarak iki ayrı yorumlayıcının varlığı, modellerin yazıldığı dillerin farklı olduğunu göstermektedir. Diller farklılaşırken anlamın aynı kalması, diller arasında bir çeviri yapıldığının göstergesidir. Bu nedenle anlamsal simetri yerine kısaca *çeviri ilişkisi terimi* kullanılabilir.

### 6.2.3 Semantik perspektif

Semantik perspektif, bir açıdan paradigmik perspektif ile sözdizimsel perspektifin üzerine kurulmuştur. Modellerin mantıksal önermelerde bulundukları varsayımından hareket edilmektedir. Mantıksal önermede bulunmak için, bir şart hakkında konuşulacak olan nesnelere referans yapabilmek ise diğer şart referans yapılan varlıkların uydukları (önermede) belirtilecek olan ilişkilerin (dolayısıyla bu ilişkilerin tanımladığı desenin) ifade edilebilmesidir. Bu nedenle semantik perspektif, referansalar için paradigmik perspektife, desenler için de sözdizimsel perspektife bağımlıdır.

Semantik perspektiften bakıldığında "Her şey bir mantıksal teoridir.". Modelin içeriğinin işaret veya desen olması durumunda yapılmış olan tanım, söz konusu işaret veya deseni bir biçimde bir sembolle eşleştirdiği için yine bir mantıksal önerme olarak görülebilmektedir. Örneğin "Ayşe Ankaraya gitti" cümlesindeki *Ayşe* kelimesi açık bir biçimde işaret (paradigmik perspektif) olduğu halde, cümlenin bağlamı içinde *Özne=Ayşe* gibi bir mantıksal önerme ile ifade edilmesi de mümkündür. Semantik perspektif bu bakış açısını destekler, diğer yandan her dil öğesini kendisi için doğal olan perspektiften tanımlamak daha yararlı bir yaklaşımdır. Dolayısıyla Ayşe kelimesi için paradigmik perspektif, cümlenin tümü için ise semantik perspektif kullanılmalıdır.

### 6.2.4 Pragmatik perspektif

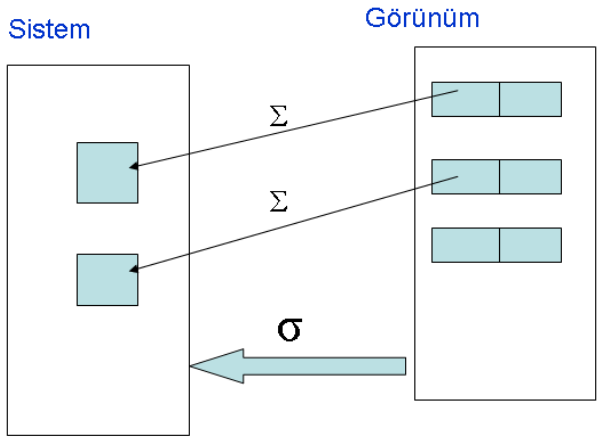
Pragmatik Perspektif'in kendi yerel birleştirme ilkesini "Her şey bir yorumlayıcıdır." veya "Her şey bir dönüşümdür" biçiminde ifade etmek mümkündür. Tüm modeller kendilerine yöneltilen sorguları, bu sorguların sonucuna dönüştüren birer dönüşüm olarak görülebilirler. Pragmatik perspektif dönüşüm kavramını destekler ve dönüşüm anlayışı üzerinden modelleme rolleri tanımlar.

Pragmatik perspektif geliştirme süreci açısından kapsayıcı bir rol oynamaktadır. Proje içindeki modeller, *arayüz modeller* kullanılarak birer *pragmatik pozisyona* dönüşürler. Bu en yalın ifadeyle modelin bir dile dönüşmesi anlamına gelmektedir. Daha dikkatli bir tanımla modelin *dinamik bir yorumlayıcı* haline geldiğini söyleyebiliriz. Bir modelin içeriği istemli bir biçimde kalıcı bir dil oluşturacak biçimde düzenlenebilir. Bunun karşıtı olarak model çok özel bir durumun modeli olabilir ve içindeki eşleştirmeler, özellikler veya önermeler tamamen bağlama özgü, geçici bilgiler olabilir. Semiyotik biliminin bir terimi olan pragmatik (pragmatics) tam olarak bu durumu karşılamaktadır. Bu çalışmada bir model bağlamı içinde aktif olan diller ve bağlamsal yorumlayıcılar (yerel bilgiler) bütününe *pragmatik pozisyon* adını vermekteyiz. Benimsediğimiz geliştirme yaklaşımında her modelin bir *pragmatik pozisyonu* vardır ve her model projenin *pragmatik pozisyonlarından* birine eklenerek onu değişikliğe uğratır. Bu ilişkiler *mega-modelleme* yoluyla düzenlenir.

### GÖRN: Görünüm ilişkisi σ

Model dönüşümü, dönüşen modelin farklı biçimlerde kopyalanması veya kodlanması biçiminde yapıldığında μ ilişkisi doğmaktadır. Modellemenin bir başka yolu da var olan bilgiye farklı bir erişim yolu yaratmaktır. Bu iki anlayış arasındaki fark, yazılım bileşenlerinin *kopyala-yapıştır* yöntemi ile yeniden kullanılması ile kalıtım yoluyla yeniden kullanılması arasındaki fark gibidir. *kopyala-yapıştır* yönteminde, (μ ilişkisine karşılık gelen) benzer (veya aynı) fakat tamamen yeni bir sistem yaratma yolu seçilmektedir. Kalıtım yönteminde ise var olan koda yeni bir başvuru yaratma yolu tercih edilmektedir.

Bu tür ilişkileri adlandırmak için Favre ve Nguyen ilişkilerinden herhangi biri uygun değildir. Bu yüzden yeni bir ilişki türü olarak σ (küçük sigma) ilişkisi tanımlanmıştır. σ ilişkisi gösterme ilişkisi Σ (büyük sigma) ile yakından ilgilidir. σ ilişkisi bir sistem ile bu sistemin görünümlerinden birini bağlar. (Şekil 6.6) Bir görünüme erişmek yoluyla sistemin kendisine dolaylı olarak erişmek mümkündür. Fakat görünüm, sistemi kendine özgü bir perspektiften gösterir ve sisteme ait bir rolü tanımlar.




Şekil 6.6. Görünüm (σ) ilişkisi

Görünüm ilişkisi ile bağlanan model aslında bir dönüşüm rolündedir ve sembolik işaretleri gerçek başvurulara çevirmektedir. Görünüm ilişkisi model dönüşümlerinin ilk aşamasında meydana gelen eşleştirme (mapping) işlemin sonucunda ortaya çıkmaktadır.

### CRRS: Corresponds to ilişkisi

Corresponds to ilişkisi mapping oluşturmayı sağlar. Dönüşümlerin en alt seviyede doğrudan yazılması bu ilişki ile mümkün olur. Mümkün olan her bir değer için karşılık gelen değerin belirtilmesi yoluyla dönüşüm yazılması en temel ve basit dönüşüm tanımlama biçimidir.

CRRS(Thing1, Thing2)

Parametreler: Üzerinde herhangi kısıt olmayan iki parametre

### TRAN: Translator ilişkisi

Translator rolü. İki dil arasında çeviri yapan bir yorumlayıcıyı bu iki dile bağlar.

TRAN(Translator, Lang1, Lang2)

Parametreler: Translator: Çeviri yorumlayıcısı; Lang1: Birinci dili gösteren bir işaret; Lang2: İkinci dili gösteren bir işaret

### ENCD: Encoder ilişkisi

Bir modelin bir düğümünü gösteren bir işareti verilen bir dilde oluşturan bir dönüşümdür. Bu durumda model, alanı modelleme rolündedir. Modelin elemanları (düğümleri) ise alanın meşru değerlerini oluşturur. Örneğim model hafta'yı modelliyorsa, ENCD ilişkisi haftanın günlerini verilen bir dilde yazabilen bir dönüşümü hafta ile eşleştirir.

ENCD(İşaretDili, Model)

Parametreler: İşaretDili: Sembolik işaretin ifade edileceği dil, Model: Dönüşümün girdisinin elemanlarına işaret ettiği (elemanlarından birine bir dizinsel işaret olduğu) model.

### DECD: Decoder ilişkisi

Decoder rolü. Bir dilde yazılmış bir ifadeyi model üzerinde yer alan bir düğümün adresine dönüştürür. (Daha önce σ işareti ile gösterilmiştir.) Uygulamalarda karşılaşılan color picker veya file picker gibi kullanıcı arayüzleri (genellikle dialog box biçiminde) bu tip dönüştürücülerin örnekleridir. (Kullanıcı girdisini belli bir uzay üzerinde bir noktayı işaret eden bir işaretçiye dönüştürdükleri için)

DECD(İşaretDili, Model)

Parametreler: İşaretDili: Sembolik işaretin ifade edildiği dil, Model: Elemanlarına işaret edilecek olan (elemanlarını gösteren dizinsel işaret döndürülecek) model.

### CONT: Controller

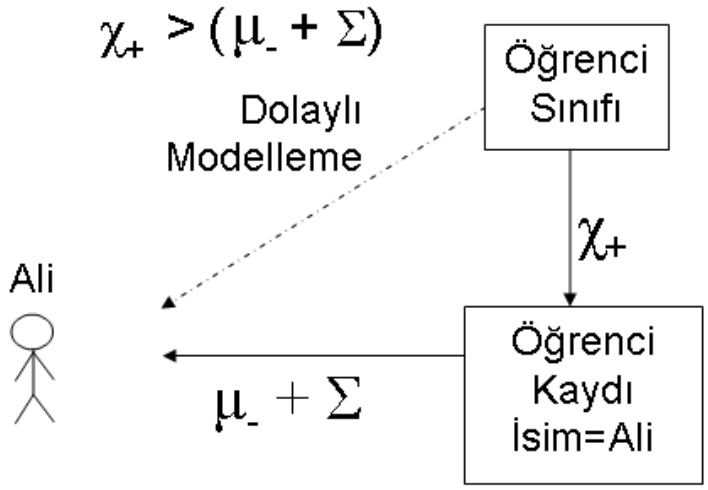
Controller rolü. Aldığı mesajlara bir tepki olarak verilen bir model üzerinde değişiklik yapar.

CONT(ControllerM, ObjectM)

Parametreler: ControllerM: Kontrol eden dönüşüm (yorumlayıcı), ObjectM. Kontrol edilen model.

### 6.2.5 İlişkilerin birlikte kullanılması

Üst modeller eksik olduklarında sistemleri yaratmak için kullanılamazlar fakat sistemleri çözümlemek (bir başka deyişle okumak) amacıyla kullanılabilirler. Şekilde böyle bir okuma işlemi şema olarak gösterilmiştir. $\tau_{oku}$ ilişkisi bir okuma dönüşümü göstermektedir. Okuma dönüşümü girdi olarak bir sistem ve üst-model kullanmaktadır ve çıktı olarak de sistemin bir görünümünü (view) oluşturmaktadır. Görünüm yukarıda söz edildiği gibi sistemle σ ilişkisi içindedir. Üst-model eksik olduğu için görünüm de eksiktir (ve eksi işareti taşımaktadır). Ayrıca görünüm ilişkisini üst model adı ile etiketleyerek daha açıklayıcı bir ifade elde edilmektedir (σ 'nın üstünde yazan U).




Şekil 6.7 Varlıkların yazılım tarafından modellenmesi.

Şekil 6.7'de genişletilmiş mega-ilişkisel dili farklı bir örnek üzerinde görülmektedir. Her şey bir modeldir ilkesi (Bezivin, 2005) sadece yazılım yapıtlarını değil, tüm bilgi işlem alanlarını kapsamaktadır ve veriler de bunun içindedir. Bellekteki bir öğrenci kaydı, gerçek dünyadaki bir öğrencinin bir modelidir ve kayıt ile öğrenci arasındaki ilişki $\mu_-$ ile gösterilir ve eksik çeviri olarak adlandırılmaktadır. Fakat kayıt sadece simetrik modelleme yapmamakta, aynı zamanda öğrencinin adı soyadı ve numarası gibi haneler aracılığıyla modellediği nesneyi işaret de etmektedir. Bu nedenle aradaki ilişki $\mu_- + \Sigma$ şeklinde gösterilmiştir. Öğrenci kaydı ile öğrenci sınıfı arasında ise $\chi_+$ ilişkisi (tam uygunluk ilişkisi) vardır. Sınıf, nesne için bir yapısal şablon oluşturmaktadır ve onu yaratacak kadar bilgi içermektedir.

Yaklaşımımızın bir başka özelliği de var olan modelleme ilişkilerinden hareketle bunların bileşkelerinin de kodlanabilmesidir. Şekildeki sınıf ile gerçek öğrenci arasındaki dolaylı modelleme ilişkisi var olan bağlantılar üzerinden χ+ > (μ- + Σ) biçiminde yazılmaktadır.

### 6.3 Mega-modellemenin Uygulanması

UML benzeri diller kullanılarak yürütülen nesneye yönelik yazılım geliştirme süreçleri gözlemlendiğinde, *sınıf diyagramı* olarak bilinen formatın gerçekte farklı amaçlarla kullanılabildiği görülür. Sınıf diyagramlarının öncelikli iki görevi, analiz safhasında, *alan modelleme* görevinde kullanılması, tasarım safhasında ise *tasarım sınıf diyagramı* olarak işlev görmesidir. Bunların dışında, framework'ler, kütüphaneler ve tasarım desenlerinin belgelenmesi gibi yeniden kullanıma yönelik kullanımlar vardır. Model Güdümlü Mimari kapsamında (veya ona öykünen teknolojilerde) UML dilinin sınıf diyagramları, meta-modelleme rolünü de üstlenmiştir. Ayrıca *Platformdan Bağımsız Modelleme* (PIM) ve *Platforma Özgü Modelleme* gibi farklı kullanımlar ortaya çıkmıştır.

UML'in başarılı yaygın kullanımı herhangi bir kuramsal sınır ile karşılaşmış değildir ve yukarıda sayılan örneklerin, herhangi bir geliştirici tarafından doğal bir biçimde çoğaltılması ve yeni kullanım alanları bulunması mümkündür. UML sınıf diyagramları bir sistemin herhangi bir bölümünü, belli bir özelliğini veya yönünü (aspect) modellemede kullanılabileceği gibi, sistem üzerindeki değişikliklerin tasarlanmasında veya belgelenmesinde de kullanılabilir.

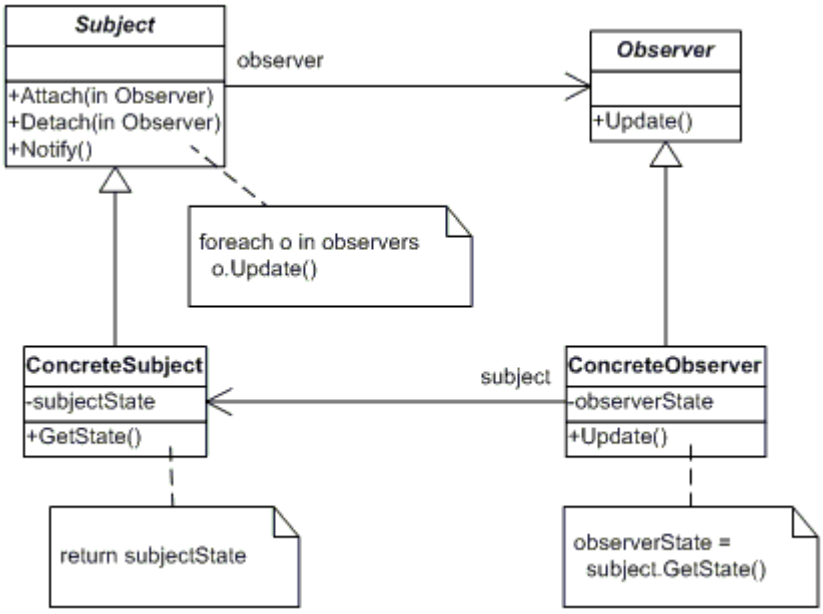
Model güdümlü kullanımda da Platform Bağımsız Modelleme ile Platforma Özgü Modelleme arasında birçok ara katmanın olması doğaldır. Platform kavramı bilgi işlem dünyasında, somut araçların yanı sıra, bunların bir silsile içinde dayandığı bir dizi soyut paradigma ve standardı da kapsamaktadır. Örneğin Oracle veri tabanı yönetim sistemi üzerinde geliştirilmiş bir uygulamanın Platformdan Bağımsız Modelinin tam olarak hangi platformdan bağımsız olduğunu söylemek zordur. Model sadece Oracle markasından bağımsız olabileceği gibi, SQL dilinden, ilişkisel veri tabanı teknolojisinden veya tümden veri tabanı kullanımından bağımsız olabilir. Günümüz projelerinde kullanılan araçların ve teknolojilerin çokluğu, ve teknolojilerin birbirlerine dayanma veya birbirlerini genişletme ilişkilerinin karmaşıklığı göz önüne alındığında, bu basit örnekte ifade edilen durumun çok daha zorlu karmaşalar yaratmaya aday olduğu görülmektedir.

Platform kavramının bulanıklığı, modeller için platforma özgü ve platformdan bağımsız gibi iki temel sınıf yerleştirmek yerine, genel bir sınıflandırma stratejisi geliştirilmesini gerekli kılmaktadır. Daha yararlı bir bakış açısı tüm modellerin gerçekte bazı platformlara özgü, ve bazı platformlardan bağımsız oldukları noktasından hareket etmelidir.

Tez çalışmasında belirlediğimiz yaklaşımda BLUE-M modelleri birer dilsel bileşen olarak kabul edilirler ve her model en az bir dilin parçası olmak üzere oluşturulur.

### 6.3.1 Tasarım deseni mega-modelleme örneği

Şekil 6.8'de yaygınca bilinen Observer Deseninin (Gamma et al., 1994) UML gösterimi görülmektedir. Şekildeki yapının bir benzerini BLUE-M şeması olarak yaratmak da mümkündür.

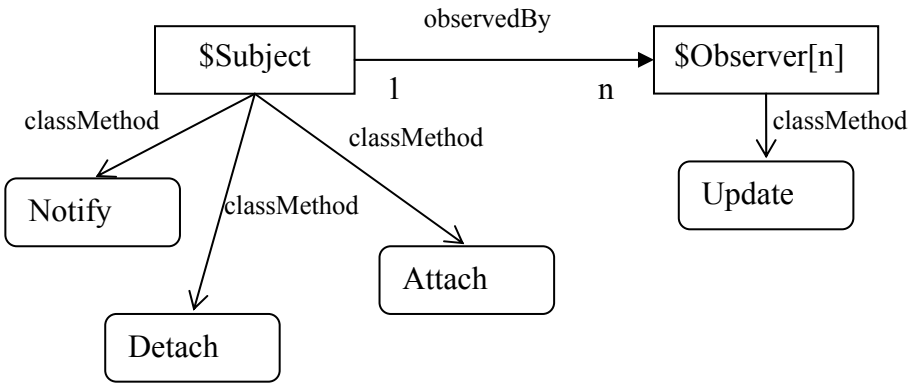



Şekil 6.8 Observer Deseni

Tasarım deseninin UML modelinde, yapıtın oynadığı desen rolüne ilişkin bir bilgi yer almamaktadır. Desen herhangi bir projenin herhangi bir tasarım modeli gibi tanımlanmıştır. Diğer yandan tasarım desenleri, sıradan tasarımlardan farklı olarak, tasarımlar için şablon oluşturma özelliğine sahiptirler. Daha somut bir ifadeyle; tasarımların kod üretmesine benzer şekilde tasarım desenleri tasarım üretirler. Desenin elektronik olarak kullanılabilir olması için bu bilginin kullanılan yazılım geliştirme aracının yorumlayabileceği biçimde kodlanması gerekmektedir.

Observer deseninin üst bölümü ile alt bölümü soyutluk derecesi açısından farklı katmanlara aittir. Şekil 6.8'de yukarıdan aşağı inen paralel kalıtım ilişkileri aslında bu soyutlama sınırını aşmaktadır ve asıl desen olan üst bölümün (Subject ve Observer sınıfları) kullanımını göstermektedir. Bu biçimi ile Observer tasarım deseni kendi içinde bir çeşit mega-modelini de içermektedir. Benzer bir durum *Factory Method* ve *Visitor* gibi bazı başka tasarım desenlerinde de gözlenmektedir.

BLUE-M bakış açısından ise tasarımın şablonu ile tasarımın kendisi iki ayrı linguistik birimdir, ve dolayısıyla asıl yeniden kullanılabilir bölüm şemanın üst bölümüdür. Bunun yanında alt bölümün nasıl üretildiğini gösteren bir *arayüz model* gerekmektedir. Şekil 6.9'da *Observer* tasarım deseninin sadeleştirilmiş biçimi gösterilmektedir.

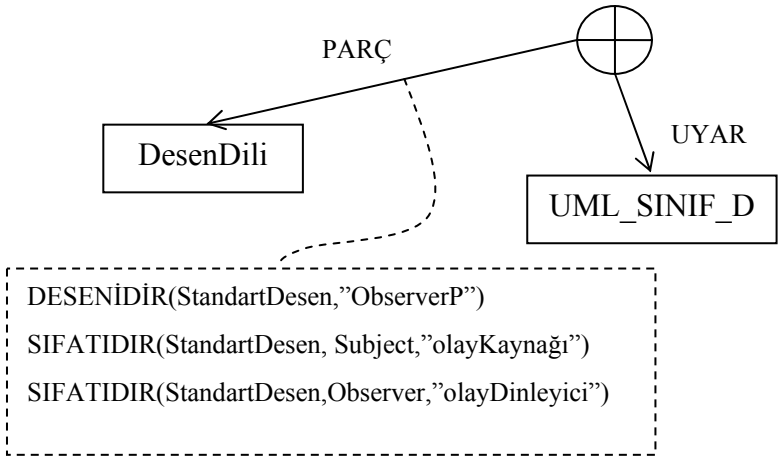



Şekil 6.9 BLUE-M içinde Observer Deseni

Şekilde görülen şema bir BLUE-M modelidir ve Observer tasarım deseninin eşdeğeri olacak biçimde işlev görebilmektedir. Şekilde *Subject* ve *Observer* sembollerinin başındaki "$" işareti, bunların birer değişken olduğunu göstermektedir. Metot isimlerinin içinde bulunduğu kutucukların yuvarlatılmış olması bu elemanların içsel detaylara sahip olduğunu göstermektedir. Bunları tıklayarak her birinin iç yapısını gösteren şemalar görüntülenebilir.

Tasarım desenini örneklemek amacıyla kalıtım yerine doğrudan BLUE-M 'in desen örnekleme yeteneği kullanılır. Bunun yapılabilmesi için bu yapının ortam içindeki geçerli bir dilin deseni olarak tanıtılması gereklidir. Verilen bir modeli bir desen olarak tanıtmak için iki yol izlenebilmektedir. Bunlardan

birincisi ifadenin bir desen olarak sahip olması gereken esnekliği sağlamak amacıyla içerdiği değişkenlerin ifade içinde belirlenmesidir. Diğer yöntem de desen işlevini yerine getirmesini sağlayacak biçimde okuma ve yazma dönüşümlerinin yazılmasıdır.




Şekil 6.10 Observer Deseni'nin arayüz modeli

Şekil 6.10'da görülen arayüz modeli birinci yolu izlemektedir. *Arayüz modelleri* proje mega-modelinin tek bir modeli ilgilendiren bölümüdür ve bir model için *arayüz modeli* oluşturulduğunda güdülen amaç, bu modelin içeriğinin anlamlandırılması için başvurulacak olan bağlamı tanımlamaktır. Bir model için birden fazla arayüz modeli yazılabilir ve farklı projeler için farklı roller tanımlanabilir.

*Arayüz modeli*, mega-model gibi asıl modeli bir düğüm olarak içinde bulundurur ve asıl modele doğrudan bağlı olduğu için ona ayrıca referans yapmasına gerek yoktur. Asıl model şekildeki gibi içinde istavroz biçimli çizgilerin olduğu bir daire ile gösterilir.

Örnekte model, *DesenDili* olarak adlandırılmış bir dilin parçası olarak tanımlanmıştır. Arayüz şemasında detay gösterilmese de, ilişkinin detaylı yapısı şematik olarak veya bir metinsel yönerge olarak incelenebilir. Şekil 6.10'da bu içerik kesikli dikdörtgen içinde gösterilmiştir. Şekildeki *DESENİDİR* ilişkisi, tanımlanmış olan modeli, *DesenDili*'nin, *ObserverP* isminde desen rolündeki bir bileşeni haline getirmektedir. Bu ilişki iki parametre ile tanımlanmıştır. *StandartDesen* olarak verilmiş olan birinci parametre, modelin hangi değişkenlerinin desene ait değişkenler olacağını belirleyen bir liste ifadesidir ve

burada bu listeyi döndüren *StandartDesen* ismindeki dönüşüm kullanılmıştır ve modelin tüm değişkenlerini döndürmektedir. İkinci parametre ise tanımlanan desenin ekleneceği dil içindeki adını belirlemektedir.

Modelde ayrıca iki *sıfat rolü* tanımlanmıştır. *Sıfat rolü*, *desen rolü*'nün özel bir durumudur. Desen kavramı, tüm yapının isimlendirilmesini sağlarken (*ObserverP* olarak), *sıfat rolü*, desenden hareketle sadece bir düğümün isimlendirilmesini sağlamaktadır. Bir başka deyişle, desenin varlığını sadece belirli bir düğümün özelliğiymiş gibi tanımlamayı sağlamaktadır. Örnekte bu imkandan yararlanarak *olayKaynağı* ve *olayDinleyici* rolleri tanımlanmıştır.

Desenin ham bir model üzerinde aranıp bulunması söz konusu olduğunda *bir ObserverP örneği bul* veya *bir olayKaynağı bul* biçiminde verilecek olan komutlara, (buradaki tanımın özdeşliği dolayısıyla) ortamın benzer tepkiler vermesi gerekmektedir. Diğer yandan Observer Tasarım Desenin var olan sınıflar üzerinde gerçekleştirilmesi söz konusu olduğunda, sıfat rolü daha anlamlı olmaktadır. Bu durumda *myClass sınıfını olayKaynağı haline getir* içeriğindeki bir komutun ortam tarafından yorumlanması bu tanımlar sayesinde mümkün olmaktadır.

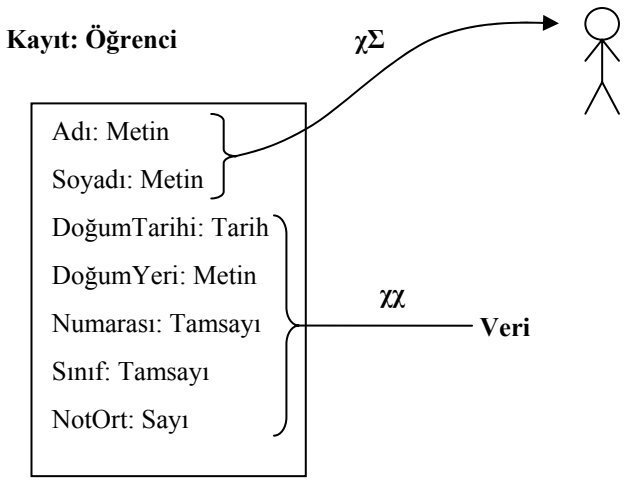
Arayüz modeli, bir modelin sadece sağladığı modelleme servislerini değil aynı zamanda kullandığı servisleri de belirlemede kullanılmaktadır. Böylece platform bağımlı veya platform bağımsız model olarak sınıflandırmanın ötesinde tam olarak hangi platforma nasıl bağlı olduğunun tanımı yapılabilmektedir. Örnekteki *UYAR* ($\chi$) ilişkisi, modelin UML sınıf şeması soyut sözdizimine uyduğunu bildirmek amacıyla konmuştur. Ortam içinde daha önceden tanımlanmış ve proje ile birlikte yüklenmiş bir UML (soyut) sözdizimini bir desen olarak tanımlamaktadır ve model de bu sözdizimine uymaktadır. Bu ilişki sayesinde modelin mantıksal açıdan UML elemanı olan birimleri, daha önce UML için yazılmış olan yorumlayıcılardan yararlanabilecektir.

### 6.3.2 Veri mega-modelleme örneği

Tüm modeller gibi veri yapılarının da semiyotik mega-modellemesi yapılarak ortama entegre edilmesi mümkündür. Tasarım desenleri soyutluk yelpazesinin en soyut ucuna yakın bir noktaya konumlandırılabilirken, veriler diğer aşırı uçta yer alırlar. Verilerin ortama entegre edilmesi ancak, sadece geliştirme ortamı olarak değil, çalıştırma ortamı olarak da BLUE-M'e dayanan

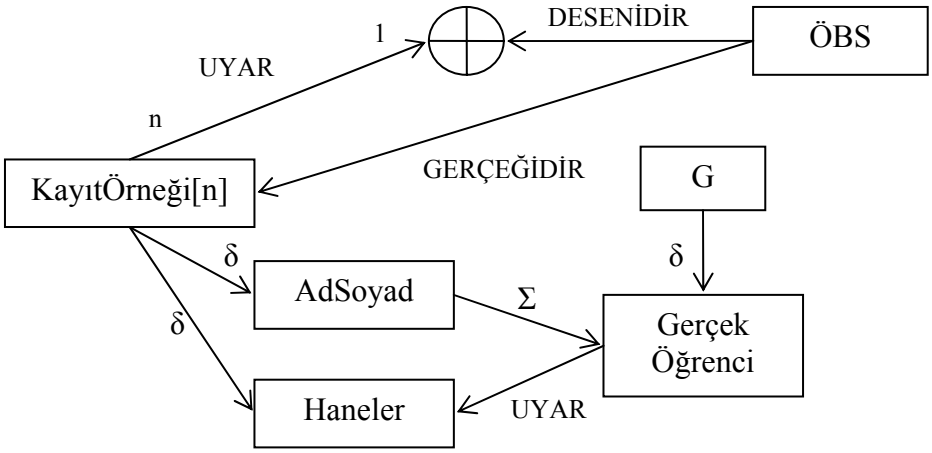
projeler için mümkün olabilir ve bunlar yapay zeka odaklı projeler olacaktır. Verilerin model olarak görülmesi ve dolayısıyla projenin geri kalanı ile aynı dilin bir parçası olması klasik programlama görevlerine kıyasla daha esnek bir sorun çözme anlayışını desteklemektedir. Prolog ve LISP gibi yapay zeka odaklı diller tarafından, *mantıksal programlama* ve *fonksiyonel programlama* paradigmalarının yüksek saflıkta uygulanmaları bu tür bir esneklik sağlamaya yönelik bir tercihtir. *Birleştirme ilkesinin* benzer bir esnekliği model güdümlü yazılım geliştirme için sağlaması gerekmektedir.

Şekil 6.11'de bir öğrenci kaydı veri tipinin semiyotik yorumlarından biri şematik olarak gösterilmiştir. Bu yorum verinin gerçek dünyadaki anlamını kodlamaya yöneliktir ve veri yapısını bir mantıksal önermenin deseni olarak değerlendirmektedir. Bir başka deyişle *öğrenci kaydı*'nın bu yorumuna göre bu veri tipi gerçek bir öğrenci hakkında mantıksal önermede bulunmak için bir desen (şablon) sunmaktadır.




Şekil 6.11 Kayıt yapısının yorumlanması

Öğrenci kaydı deseni, (desen terimi burada BLUE-M bağlamında kullanılmıştır) tanımladığı önermelerin simetrik bir modelidir. Mantıksal önermenin yapılması için gereken referans ve ilişkisel desenlerin, desenleri bu kayıt deseni içinde yer almaktadır. Şekil 6.12'de öğrenci kaydı'nın bir mega-modeli gösterilmektedir. Şekilde bazı ilişkiler yerine Yunan harfi kısaltmalar kullanılmıştır. Ayrıca *gerçek dünya* pragmatik pozisyonunu belirtmek amacıyla BLUE-M içinde önceden tanımlı bulunan "G" sabiti kullanılmıştır.




Şekil 6.12 Öğrenci kaydı yapısının bir arayüz modeli

Gerçek dünya açısından kaydın Ad ve Soyad'dan oluşan ilk bölümü örneklendiğinde bir işaret elde edilir (öğrencinin ismi). Bu işaret gerçek dünyada kaydı tutulan asıl öğrenciyi işaret etmektedir. Gerçek dünyada işlev gören bir işaretçi sadece son derece soyut bir düzeyde programlanmış hayali bir kullanıcı arayüzünde doğrudan referans amacıyla kullanılabilir. Bu arayüz örneğin doğal dil tabanlı bir kullanıcı arayüzü olabilir. Bunun dışında gerçek dünya referansları mega modeller için bir tür çıkmaz sokak veya yaprak-düğüm olarak görülebilirler.

Anlamlandırma açısından ise gerçek dünya referanslarının çapa (anchor) olarak önemli bir işlevi vardır. Anlamı bilinen bir nesne ile kurduğu ilişki üzerinden, bazı başka nesnelerin anlamları tanımlanabilir. Bu açıdan bakıldığında tamamlanmış gerçek bir anlamsal tanımın sadece gerçek dünya nesnelerine bir yoldan bağlanan bir mega-modelde söz konusu olabileceği, ne kadar karmaşık olursa olsun kapalı ve döngüsel tanımların sözdizimsel düzeyde kalacağı görülebilir.

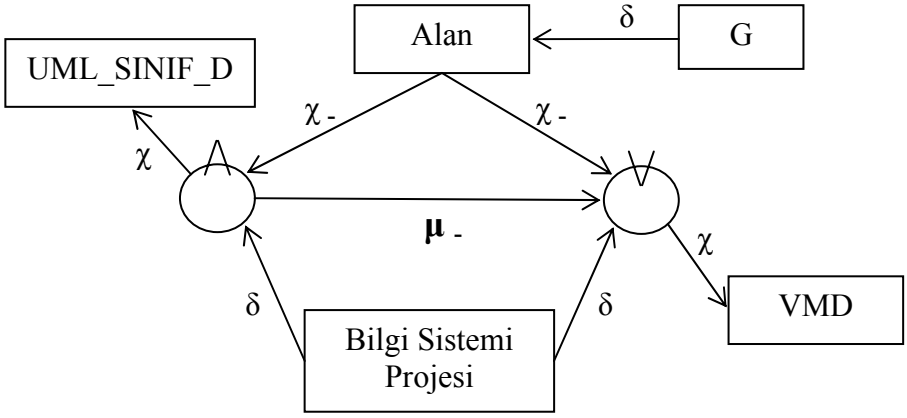
Örnek model *ÖBS: Öğrenci Bilgi Sistemi* adı verilen ve öğrenci veritabanının içerdiği tüm bilgileri kendi yorumlama fonksiyonu olarak kullanan hayali bir dile iliştirilmiştir. Modelin kendisi dilin bir deseni olarak tanımlanmıştır. Asıl modelin (kayıt yapısının) formatına bağlı olarak yazılan bir dönüşüm, gerekli desen yorumunu yapabilir (mega modelde dönüşüm ismi belirtilmemiştir). Modelin bir örneği olarak tanımlanmış olan (*UYAR* ilişkisiyle bağlı) *KayıtÖrneği* elemanı ise *ÖBS* dilinin bir gerçeği olarak tanımlanmıştır. Bu ilişki, *KayıtÖrneği* nesnesinin, *ÖBS* yorumlayıcısına eklenen bir mantıksal önerme olduğunu göstermektedir. Bu önermeler *ÖBS* içinde sorgulanabilir bir

bilgi tabanı oluşturmaktadır ve diğer yapılarla birlikte *ÖBS* dilinin *pragmatik perspektifteki* özelliklerini belirlemektedir.

### 6.3.3. Dönüşüm mega-modelleme örneği

Her bir dönüşüm bir girdiye bir de çıktıya sahiptir. Dönüşümlerin mega-modellemesi, girdi ile çıktı arasındaki ilişkinin modellenmesine dayanır. Modellerin anlamlandırılması konusundaki motivasyonun dönüşümler için de devam ettirilmesi gerekmektedir. Yalnızca dönüşümlerin girdi ve çıktısı üzerinde kısıtlar tanımlamakla yetinen bir yaklaşımın yeterince anlamlı bir mega-model tanımı vermesi beklenemez. Kullanılan model iki nokta arasında en az bir köprü kurmadıkça dönüşümün kendisi hakkında herhangi bir önermede bulunduğu söylenemez.

Şekil 6.13'te yaygınca bilinen bir örnek olan nesneye yönelik model ile veri tabanı modeli arasındaki dönüşüm için bir arayüz modeli tanımlanmıştır. Şekildeki *VMD* düğümü *Veri Modelleme Dili* anlamındadır. Kaynak model, içinden bir ok başı çıkan bir daire ile, hedef model ise içine bir ok başı giren bir daire ile gösterilmiştir.




Şekil 6.13 Dönüşüm için arayüz modeli örneği

Kaynak ile hedef arasındaki temel ilişki $\mu_{-}$ ile gösterilmiştir ve bu *eksik çeviri* (veya eksik simetri) olarak adlandırmış olduğumuz ilişkidir. Kaynaktan hedefe gitmek için yeterli bilgi olmadığını göstermektedir. Veri tabanı modeli, nesneye yönelik modelin içerdiği bilgilere ek olarak indeks üretmekte kullanılacak olan anahtar alanların listesini gerektirmektedir. Şemada görülen bir başka bilgi de, hem kaynak hem de hedef modellerin, gerçek dünyaya ait bir (aynı) alanın *eksik uygunluk* tipinde modelleri olduğudur. Aslında kaynak ile hedef arasındaki simetrinin nedeni bu ortak ilişkidir. *Alan* her iki modele de farklı ve eksik bir

açıdan uymaktadır ve kaynak ve hedef modellerden hiçbiri *Alan*'ın tam bir modellemesini yapamamaktadır. Ne yazık ki *eksik uygunluk* ilişkisi tek yönlü bir ilişkidir ve bu köprü üzerinden dönüşümün otomatik üretimi mümkün değildir.

Şemanın alt ucunda bulunan Bilgi Sistemi Projesi düğümü yürütülmekte olan projeye karşılık gelen bütüncül modeli temsil etmektedir. δ (içerme) ilişkisinin detayı şemada gösterilmemekle beraber kaynak ve hedefin, aynı projenin biri nesneye yönelik bölümünü diğeri veri tabanı bölümünü temsil ettikleri, ilişkilerin detaylı modelinde belirtilmektedir

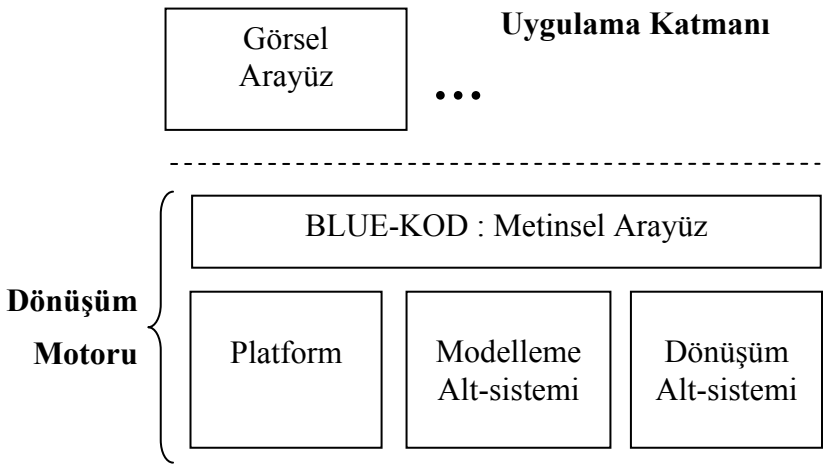
# 7. DÖNÜŞÜM DİLİ TASARIMI

Model dönüşüm dili projemiz *Basic Linguistic Unification Environment for Modeling* (Modeller için Basit Dilsel Birleştirme Ortamı) tamlamasının bir kısaltması olan "BLUE-M" ismini almıştır. İsim aynı zamanda *Blue Morpho* isimli bir kelebek türüne atıfta bulunmaktadır.

BLUE-M dilinin tasarımı kapsayıcı bir model ve dönüşüm anlayışından yola çıkmakta ve geliştiriciye dönüşümler üzerinde serbestlik ve yüksek bir kontrol sunmayı hedeflemektedir. Ayrıca modellerin ve dönüşümlerin, makineler tarafından algılanabilecek biçimde anlamlandırılması hedeflenmiştir. Böylece otomasyona ve yeniden kullanıma uygun bir zemin oluşturulması umulmaktadır.

## 7.1 Genel Mimari

Şekil 7.1'de BLUE-M model dönüşüm platformunun genişletilebilir mimarisi gösterilmektedir. En alt katmanı oluşturan alt-sistemler, modelleme ve model dönüşümü işlevlerini desteklerler. Bunun yanında model kavramının çok genel bir yorumunun benimsenmiş olması ve model dönüşümünde, geliştirici tarafından kodlanabilecek her türlü dönüşüme izin verilmesi nedeniyle, dönüşümlerin kodlanmasında, programlama düzeyinde kontrole izin veren bir dönüşüm anlayışı gerekli olmuştur. Bu nedenle klasik programlama kavramları olan değişkenler, bağlam ve akış gibi öğeleri destekleyen bir platform bölümüne ihtiyaç olmuştur.




Şekil 7.1. Model Dönüşüm Dili Mimarisi

Alt sistemler tarafından gerçekleştirilen modelleme ve dönüşüm işlevleri, bir üst katmanı oluşturulan ve metin tabanlı bir dil olan BLUE-KOD dili tarafından toplanır ve, üst katmana sunulur. Bu dili kullanarak dönüşüm motoruna kumanda etmek mümkündür.

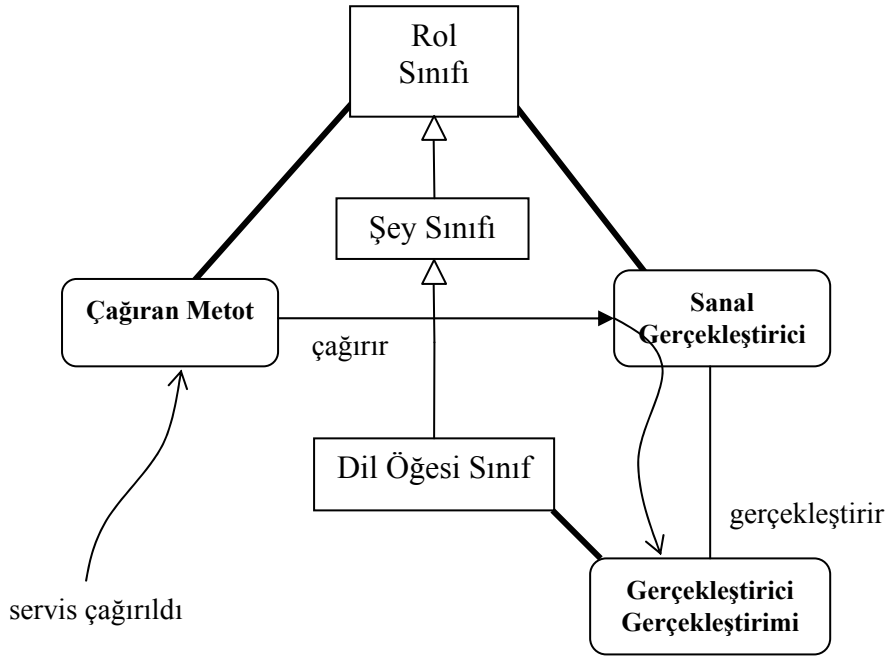
Çizelgede dil öğelerinin okuma, yazma ve çalıştırma görevlerine ilişkin olarak oluşturulmuş olan rollerin elde edildiği, işlevler ve rol sınıfları arasındaki ikili eşleşmeler gösterilmiştir. Dönüşüm işletme görevlerinin üç işlevi *okuma*, *yazma* ve *işletme*, tablonun dikey değişkenini oluşturmaktadır. Yatay eksende ise, söz konusu işlevlere ilişkin oynanması gereken (sanal) rol sınıfları görünmektedir. *Aktif, pasif* ve *mesaj* rol olmak üzere üç değişik sınıfta rol vardır. Bir işleve ilişkin *aktif* rol, o işlevi yerine getiren özneyi; *pasif* rol, o işlevden etkilenen veya işlevin hedefi olan nesneyi, *mesaj* rolü ise özne ile nesne arasında taşıyıcı rol oynayan yapıyı göstermektedir. İşlevler ile rol sınıflarının kesişimlerini gösteren tablo gövdesinde ise tanımlanmış olan dokuz dönüşüm rolüne karşılık gelen soyut sınıflar gösterilmiştir. Bunlardan *OkuMesaj* ve *YazMesaj* sınıfları, okuma ve yazma işlevlerinde, kendine özgü mesaj davranışına gerek görülmediği için gerçekleştirilmemiştir. Diğer soyut sınıflar ise kodlanmıştır. Bu rolleri gerçekleştirme veya atlama kararı, dile eklenen her bir dil öğesi için alınması gereken bir karar olmaktadır. Örneğin verilen bir dil öğesinden herhangi bir değerin okunmasının mümkün olup olmayacağına veya bu öğenin çalıştırılabilir bir öğe olup olmadığına karar vermek gerekecektir. Bunun dışında okunabilirlik veya çalıştırılabilirlik gibi özellikleri dinamik olarak açıp kapatmak da mümkündür.

Çizelge 7.1 Roller ve Soyut Sınıflar

| | **Aktif** | **Pasif** | **Mesaj** |
|---|---|---|---|
| **Okuma** | Okuyan | Okunan | OkuMesaj |
| **Yazma** | Yazan | Yazılan | YazMesaj |
| **Çalıştırma** | Çağıran | Görevli | İş |

Dil sınıfları sıradüzeninin tepesinde yer alan PlastThing sınıfı içinde tanımlanmış olan bir dizi mantıksal değişken bu rollerin çalışma iznini düzenler ve çalışma sırasında bunları manipüle ederek rolleri açıp kapatmak mümkün olur. Bu imkan, bir çalışma-zamanı (dönüşüm-zamanı) çok yapılılık anlamına gelmektedir.

Şekil 7.2'de tasarım bir başka perspektiften görülmektedir. Burada da metotları şema içinde düğümler olarak gösterme amacına uygun olarak UML sözdizimi aşılmıştır.




Şekil 7.2. Rollerin çokbiçimli sınıflara kodlanması için kullanılan desen

Şekil 7.2'de her bir dönüşüm rolü için kodlanan sınıf ve metotların oluşturduğu yapıyı anlatan bir desen görülmektedir. Bu desen her bir rol için bir kere gerçekleştirilmiştir. Şemada sınıflar köşeli, metotlar ise yuvarlatılmış kutucuklarla gösterilmiştir. Ayrıca sınıflar sahip oldukları metotlara kalın çizgilerle bağlanmışlardır. Bu desende yer alan *Rol Sınıfı* olarak anılan sınıf ve *Çağıran Metot*, *Sanal Gerçekleştirici* ve *Gerçekleştirici Gerçekleştirimi* olarak etiketlenmiş bulunan metotlar her bir rol için ayrı bir isimle oluşturulmuştur. Rolün karşılığı (temsilcisi) olan sınıfın biri çağırılan, diğeri de üstüne yazılan (override edilen) (virtual) iki tane metodu vardır. Çağırılan metot talep edilen role izin verilip verilmediğini kontrol etmekte, veriliyorsa Virtual Gerçekleştirici'yi çağırmaktadır. Virtual metoda yapılan bu çağrı onu override eden gerçekleştirimine yönlendirilmekte ve böylece ilgili dil öğesinin kendi mantığı içinde talep edilen rolü yorumlaması ve yapılan çağrıya cevap vermesi sağlanmaktadır. Desenin kodlama safhasında metot isimleri sistematik olarak bir kurala göre belirlenmektedir.

Bu rol tabanlı tasarım, dil öğelerinin dinamik bir çokyapılılık sergilemelerini sağlayacaktır. Örneğin bir dönüşümün kritik bir noktasında bir nesneyi yazma işlemine kapatmak mümkündür. Rol tabanlı tasarımın bir başka avantajı da bir model üzerinde değişiklik yapan dönüşümler ile, bir modelden yeni bir model yaratan dönüşümler arasındaki farkı dönüşüm kuralının sorumluluk alanından soyutlayarak ayırabilmesidir. Basitçe, potansiyel olarak bir dönüşüm kuralını "değişiklik yapan kural" veya "yeni model yaratan kural" olarak özelleştirmeksizin yazmak ve her iki görevde de kullanmak kuramsal olarak mümkündür, çünkü dönüşümün çıktısının nasıl kullanılacağının sorumluluğu yazılan rolünü oynayan nesne tarafından belirlenecektir.

## 7.2 Metinsel Komut Dili: BLUE-KOD

BLUE-M geliştirme ortamı, kullanıcı arayüzü olarak taban düzeyinde kontrol sağlayan metinsel bir dile ve de onunla entegre edilmiş bir grafiksel arayüze sahiptir. Ayrıca proje her iki açıdan da gelişmeye açık bir mimari yapıyla şekillenmiştir.

Dönüşüm motorunun temel kullanıcı arayüzü, BLUE-KOD olarak adlandırılmış olan (BLUE KOmut Dili) fonksiyonel dildir. Dönüşüm motorunu kullanacak olan daha üst düzey dillerin BLUE-KOD aracılığıyla dönüşüm motoru altyapısı ile haberleşmeleri mümkündür. Dönüşüm motoru geliştirme çalışmalarının gelişimi, bu dilin gelişimi ile paralellik göstermiştir. Geliştirme çalışmalarının devamında altyapıya eklenen herhangi bir özellik veya imkân, dile eklenen yeni özellikler veya yeni komutlarla karşılanmaktadır.

BLUE-KOD dili, yorumlanan diller grubundan fonksiyonel sözdizimli bir dildir. Komut satırlarının girilebildiği bir konsol penceresi aracılığı ile veya bir metin dosyasından yüklenerek kodlama yapılmaktadır. Komutların sözdizimi LISP dilini andırmaktadır. Komut, bir parantez açma işareti ile başlamakta, sonra fonksiyon adı ve varsa ayraçlarla ayrılmış biçimde parametreler listesi gelmektedir. Aşağıda bu sözdizimi EBNF ile gösterilmiştir:

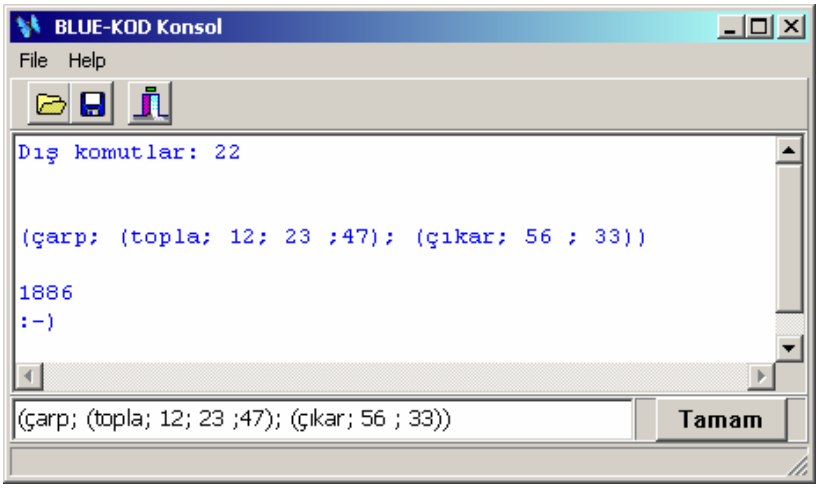
```
( fonksiyonAdı {; Parametre} )
```

Diğer fonksiyonel dillerde olduğu gibi bu çağrıların iç içe gömülmesi yoluyla daha karmaşık komutlar tasarlanabilir ve böyle bileşik bir fonksiyon

çağrısının yapısı bir ağaç biçimini alır. Örneğin (12+23+47)x(56-33) biçiminde bir aritmetik ifade aşağıdaki biçimde yazılır:

```
(çarp; (topla; 12; 23 ;47); (çıkar; 56 ; 33))
```

Şekil 7.3'de konsol penceresinde bu komutun çalıştırılması gösterilmiştir.




Şekil 7.3. BLUE-KOD konsolunda aritmetik işlem

BLUE-KOD dilinde *tamsayı*, *mantıksal*, *metinsel*, *liste* ve *çizge* veri tipleri vardır. Dilin bir parçası olan komutlarda bazı tip kontrolleri, çalışma zamanı, ortam tarafından yapılmaktadır. Kullanıcı tarafından tanımlanan modüllere yapılan çağrılarda ise herhangi bir içsel tip kontrolü yoktur.

Dilin güncel sürümünde tamsayılar üzerinde aritmetik işlemler ve karşılaştırmalar, metinsel işlemler, mantıksal işlemler, kontrol deyimleri ve temel girdi-çıktı deyimleri kullanılabilir durumdadır. Liste ve çizge veri yapıları üzerindeki işlemler ise geliştirme aşamasındadır.

Dönüşüm motorunu gerçekleştirme çalışmalarında önemli bir aşama, dönüşümlerin modüler bir biçimde tanımlanmasına olanak sağlayan yapıların gerçekleştirilmesidir. Dönüşümlerin yeniden kullanılabilir modüllere ayrılabilmesi amacıyla alt-yordamların tanımlanması ve işletilmesi düşünülmüştür. Alt-yordamlar metinlerin derlenmesi yoluyla dinamik olarak oluşturulabilirler ve çalıştırılabilirler. Aşağıdaki kod parçası, ilk satırda *çizgedenWeb* isimli bir dosyadan okuduğu metni *text* değişkenine atamaktadır.

```
(bağla;text;(oku;(mdosya;"çizgedenWeb")))
```

```
(bağla;kod;(derle;text))
(işlet;kod)
```

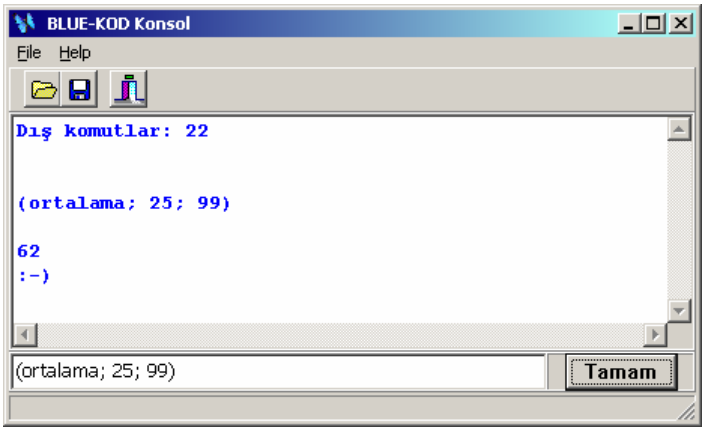
İkinci satır *text* değişkeninde saklanan metnin derlenmesini ve sonucun *kod* ismindeki değişkende saklanmasını söylemektedir. Son satırda da derlenmiş olan kod (sayısallaştırılmış komutları içeren bir listeye dönüşmüştür) çalıştırılmaktadır.Alt yordamları çalıştırmak için iki farklı yaklaşım (ve komut) vardır: *işlet* komutu bir yönergeyi global bağlamda ve dinamik bağlama (dynamic binding) kurallarıyla çalıştırır, *yişlet* komutu (yerel-işlet anlamında) ise tüm değişken isimlerini yerel bağlamda bağlar.

Dönüşüm motorunun komut ara-yüzü basit programlama işlevlerini yerine getirebilmesini sağlayan komutlarla zenginleştirilmiştir. Mantıksal veri tipi, mantıksal işlemler, karşılaştırma komutları, döngü komutu, dallanma komutu ve basit girdi çıktı sağlayan bazı komutlar eklenmiştir (bkz. Ek-I). Bunlarla birlikte alt yordamlara parametre aktarımını sağlayan ve alt yordamların değer döndürmesini sağlayan komutlar vardır.

Komut setinin programa dokunulmadan kütüphane kullanımı biçiminde, genişletilebilmesi amacıyla standart bir dizin içinde, metinsel dönüşüm yönergesi dosyaları olarak saklanan yönergelerin açılışta yüklenmesi sağlanmıştır. Bu yordamlar komut setinin bir uzantısı gibi kullanılabilmektedir. Örneğin ortalama hesaplayan bir yönerge aşağıdaki gibi iki satırlık bir kodla tanımlanabilmektedir:

```
(parametreler;p1;p2).
(döndür;(böl;(topla;p1;p2);2)).
```

Bu yönerge bir metin dosyası içinde ilgili dizine yerleştirildiğinde, ortalama komutu dilin bir parçası haline gelmektedir (Şekil 7.4).



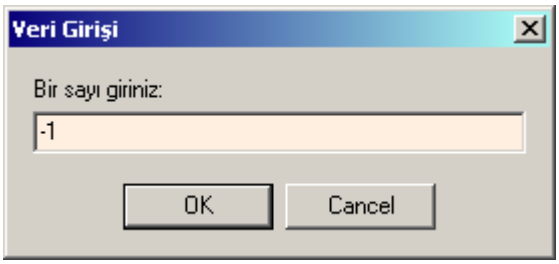

Şekil 7.4. Yönergenin yeni bir komut gibi kullanılması

BLUE-KOD yönergeleri temel programlama işlevlerini sağlayacak düzeyde girdi çıktı ve akış kontrolü komutlarını kullanabilmektedir. Aşağıda basit bir programlama uygulamasını gerçekleştiren bir yönerge örneği verilmiştir. Bu yönerge kullanıcıdan negatif bir sayı ile bitirilen bir dizi sayı girdisi almakta ve çıktı olarak girilen sayıların sayısını, ortalamasını, ve kaç tanesinin elliden küçük olduğunu vermektedir.

```
(yaz;"Merhaba!").
(yaz;"Ortalamasını bulmak için bir dizi sayı giriniz.").
(yaz;"Bitirmek için negatif sayı giriniz.").
(bağla;x;0).
(bağla;t;0).
(bağla;küç;0).
(bağla;loopkod;(koru;      //Kod bloğu
  (bağla;t;(topla;t;x));
  (bağla;x;(tamsayı;(girdi;"Bir sayı giriniz:")));
  (yaz;(metin;"%d girildi";x));
  (eğer;(ve;(küçük;x;50);(büyük;x;0));(arttır;küç))
    )
).
(döngü;(bağla;i;0);(büyükeşit;x;0);(arttır;i);loopkod).
(azalt;i).
(yaz;(metin;"%d sayı girildi";i)).
(yaz;(metin;"Ortalama:%d";(böl;t;i))).
(yaz;(metin;"%d tanesi 50 den küçük!";küç)).
```

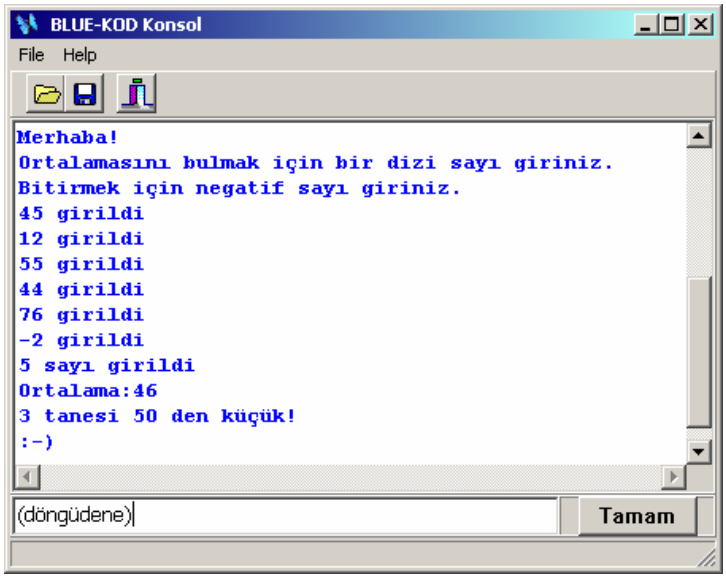
Örnekteki *yaz* komutu konsol penceresine metin yazmak için kullanılmaktadır. *Bağla* komutu atama deyimidir ve bir değişkene değer atamayı sağlamaktadır. Bu komut sıklıkla fonksiyonların döndürdüğü değerleri saklamak amacıyla kullanılmaktadır. Örnekte *koru* fonksiyonunun döndürdüğü değer *loopkod* değişkeninde saklanmıştır. Daha sonra da *döngü* komutunun son parametresi olarak kullanılmıştır. *Koru* fonksiyonu kendisine girdi olarak verilen parametreyi hiçbir değişikliğe uğratmadan aynen döndürmektedir. Bu arada yorumlayıcının çalıştırma talebine de engel olmaktadır ve LISP'deki *quote* fonksiyonuna benzemektedir. Bunun yanında, birden fazla parametre ile çağırıldığında bir sıralı liste yapısı oluşturmakta, parametreleri (yine işletmeden) bu listeye doldurmakta ve listeyi döndürmektedir.

BLUE-KOD'da sıralı listeler çalıştırıldıklarında ilk elemanlarından başlayarak tüm içeriklerini sırayla çalıştırırlar. Dolayısıyla bir kod bloğunu örnekte olduğu gibi *koru* komutu aracılığıyla bir değişkende atmak ve sonradan bu değişkenden alıp çalıştırmak mümkündür. Örnekte ayrıca akış kontrolü komutları olan *eğer* ve *döngü*'nün kullanımı görülmektedir. Bu deyimler çalıştıracakları kodu değişken içeriğinden alabilecekleri gibi doğrudan içlerine gömülü olarak da elde edebilirler. Örnekte ayrıca *girdi* olarak adlandırılmış olan basit bir diyalog komutu, kullanıcıdan veri girişi sağlamak amacıyla kullanılmıştır. Şekil 7.5'de bu diyalog kutusunun bir görüntüsü verilmiştir.




Şekil 7.5. Girdi komutu ile veri girişi

Akış kontrolü deyimlerinin yanı sıra, girdi ve çıktı komutlarının bulunması sayesinde dönüşüm sırasında kullanıcı ile etkileşime girmek mümkün olduğundan, BLUE-KOD'un desteklediği akış yapıları, katı bir dönüşüm anlayışının gerektirdiğinden daha fazla esneklik sunmaktadır. Örnekteki yönergenin konsolda çalıştırılması sonrası oluşan pencere görüntüsü aşağıdaki şekilde gösterilmektedir (Şekil 7.6):




Şekil 7.6 Yönergenin çalışması

Komut girildikten sonra mesaj görüntülenmekte ve veri girişi için diyalog penceresi ortaya çıkmaktadır. Ard arda girilen sayılar ekranda doğrulanmakta ve veri girişi negatif bir sayının girilmesi ile sonlandırılmaktadır. Sonuç olarak kaç sayının girildiği, bunların ortalaması ve kaç tanesinin 50'den küçük olduğu ekrana yazılmaktadır.

## 7.3 BLUE-M Modelleri

### 7.3.1 BLUE-M Model mimarisi

Alışıldık yazılım geliştirme süreçleri içinde, geliştiriciler tarafından modellere her başvurulduğunda, başvurulan model geliştiricinin kendi zihni tarafından, kendi amaçlarına uygun olarak anlaşılmakta ve yorumlanmaktadır. Diğer yandan makinelerin anlama ve yorumlama kapasiteleri sınırlı olduğundan, alışıldık model formatlarına nüfuz etmeleri ve çok yönlü otomasyon desteği sağlamaları mümkün olamamaktadır.

BLUE-M modelleri otomasyona doğrudan destek vermek amacıyla çalıştırılabilir sayısal nesneler olarak düşünülmüştür. Modellerin bir iş bölümü içinde birlikte çalışabilmelerinin ve birbirlerini kullanabilmelerinin, nesnelerin yaptığı gibi mesaj alışverişinde bulunmaları ile mümkün olacağı düşünülmüştür. Bu noktada model kullanımı ile model dönüşümü arasında çok yakın bir ilişki bulunduğunu söyleyebiliriz. Modeller kendilerine gönderilen mesajlar üzerinde bir dönüştürücü rolü oynamakta ve gerekli çıktıyı üretmektedirler.

BLUE-M modeli fiziksel olarak bir *listedir* ve içine yerleştirilen işaretçi üçlüleri sayesinde çizgeleri temsil edebilmektedir. İşaretçi üçlüsü, biri kaynağı biri hedefi ve biri de ilişkiyi gösteren üç işaretçiden oluşan bir veri tipidir, ve her biri çizgenin bir bağlantısını temsil etmektedir. BLUE-M işaretçileri bildik anlamda C işaretçisi (pointer) olabildiği gibi, yorumlanması gereken bir sembol de olabilmektedir. Aynı modelin içinde, değişkenler, sabitler, değer bağlanmış değişkenler, *eşleştirme (mapping)* biçiminde yazılmış dönüşüm kuralları ve desenler de bulunabilmektedir. Ayrıca model içine yerleştirilen bir bağlantı aracılığıyla başka bir modeli mesaj yorumlaması için göreve çağırmak da mümkündür. Tüm bu karma elemanlar topluluğu bir arada dinamik bir dil oluştururlar ve söz konusu modelin etkileşim biçimini şekillendirirler.

Modellerin anlamlandırılmasında modellerin kullanım biçimlerinin ön plana çıkmasının bir nedeni de birleştirme ilkesi gereği modellere ilişkin olarak benimsenmiş bulunan kapsayıcı anlayıştır. Bu anlayış gereği model kavramının yapısal bir tanımı yerine işlevsel bir tanımına ihtiyaç duyulmaktadır. Böylece basitçe çizge yapısında olmayan veya etkili bir biçimde çizge biçiminde ifade edilemeyen modelleri sisteme bağlamanın bir yolu açılmış olmaktadır. Bu fazlaca kuramsal çözümün uygulama alanındaki karşılığı sarmalama (encapsulation) olarak bilinen (nesneye yönelik programlamanın da önemli kavramlarından biri olan) tekniktir. Model tanımı olarak Minsky'nin dördüncü bölümde bahsedilen tanımı benimsenecek olursa (Minsky, 1968) (ki hemen hemen aynı tanım başka birçok filozof ve matematikçi tarafından da ifade edilmiştir), bunun uygulama alanındaki izdüşümünün bir *arayüz* (nesneye yönelik anlamıyla *interface*) olması gerekmektedir.

Bu bakış açısından model, modellediği nesne hakkında gönderilen sorgulara cevap verebilen herhangi bir birimdir. Modele gönderilen sorgunun da, çıktı olarak üretilen yapının da birer model olduğunu düşünecek olursak, yapılan işlemin aslında bir model dönüşümü olduğu söylenebilir. Dolayısıyla her model aslında bir model dönüştürücüsüdür. Şekil 7.7'de modellerin mesajlara cevap verebilmek için kullandıkları bilgilerin kaynakları gösterilmiştir. Burada görülen gerçek bir katmanlı şema değildir, ve sıralama, temel bilgi kaynakları ile karmaşık veya dolaylı bilgi kaynakları arasındadır. Gönderilen mesajın farklı yorumlarının olması durumunda hangi yoruma öncelik verileceğine ilişkin bir kriter oluşturmaktadır.


Şekil 7.7. Modellerde mesaj yorumlamada başvuru önceliği

Mesaj yorumlama açısından önceliği olan kaynaklar daha üstte yer almaktadır. Öncelikle bir modelin içinde, bir mesajı doğrudan doğruya onun yorumlanmış haliyle eşleştirebilen bir *mapping* bilgisi yer alabilmektedir (*doğrudan dönüşüm bilgisi*). Bu durumda modelin kendi fiziksel yapısına (veya fiziksel yapıda kodlanmış bilişsel içeriğine) bakılmaksızın mesaja nasıl cevap verileceği bilinmiş olmaktadır. Bu tür müdahaleler bazı modellerin bazı özelliklerini değiştirmek amacıyla yapılabileceği gibi tamamen kullanıma kapatmak amacıyla da yapılabilir.

Bunun yanında, ihtiyaç duyulduğunda bir modelde sadece *doğrudan dönüşüm bilgisi* kullanılarak küçük çaplı bir dönüştürücü oluşturmak da mümkündür. Örneğin isim uzayları bu tip modellerdir. Bir programın içinden bir değişken ismi kullanıldığında, isim uzayı bunu o değişkenin değerine dönüştürmektedir.

Modellerin ikinci bilgi kaynağı fiziksel olarak içinde barındırdığı düğümler ve ilişkilerin sergilediği yapıdır. Bu yapı, tercih edilen model anlayışının bir gereği olarak sorgulanabilir bir yapıdır. Çizgeler bağlantıları üzerinden çeşitli kısıtlarla sorgulanır ve bir görünüm (view) oluşturulur.

Üçüncü bilgi kaynağı, çizgeleri oluşturan elemanların referanslar olarak değil semboller olarak ele alınmasını gerektirmektedir. Bu durumda sorgunun döndürdüğü nesne, var olan model üzerinde bir referans veya görünüm değil, model elemanlarının işaret ettiği bir değerdir. Bu kullanımın biraz daha karmaşık bir uygulaması modelin bir bölümünü kopyalamak biçiminde gerçekleşmektedir.

*Dış başvuru* olarak anılmış olan dördüncü kaynak gerçekte modelin bir parçası değildir. Başvurulmak istenen modelin var olmaması veya erişilebilir olmaması durumunda istenen servisi verebilecek bir başka modelin kullanılmasını içermektedir. Bu ikinci model bir nedenden ötürü istenilen modelin modeli olarak görülebilecek bir başka şema olmaktadır. Bu yardımcı modelin bulunması ve kullanılması yararlı bir otomasyon örneğidir ve modelin kurduğu mega-ilişkiler ile ilgilidir.

BLUE-M görsel model ve dönüşüm geliştirme aracı, iki temel model tipine dayanmaktadır.

1. Temel model

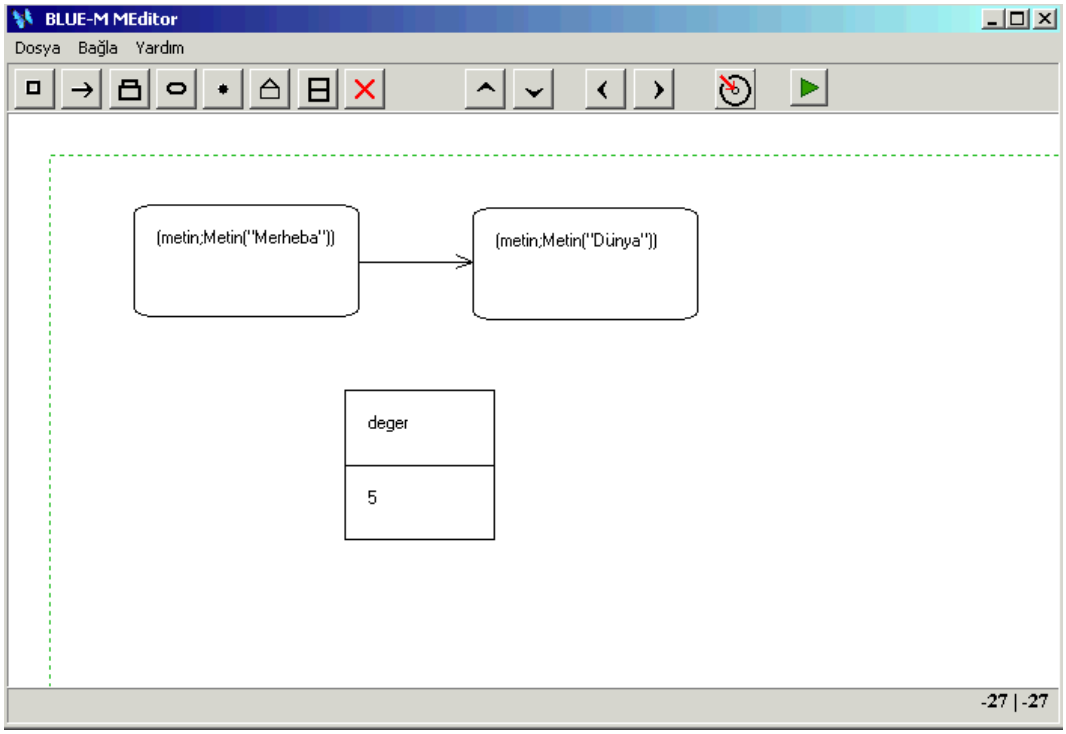
2. Akış modeli

Farklı dillerde veya görünümlerdeki modellerin giriş çıkışına ilişkin arayüzlerin oluşturulması, esas olarak dönüşüm biçiminde yazılması gereken eklentilere bırakılmıştır. Geliştirme ortamı hakkında belirtilmesi gereken bir başka özellik de, bir projenin yürütülmesi sırasında üzerinde çalışılan farklı modellerin, gerçekte birbirleriyle yakın ilişkide olmalarıdır. Tüm modelleri gerçekte tek bir büyük modelin parçaları veya görünümleri olarak görmek gerekmektedir.

### 7.3.2 Model editörü

Temel BLUE-M modeli basit bir yapıdadır ve diğer model türlerinin atası durumundadır. Düğüm ve ilişki gibi jenerik elemanlar içerdiği için dönüştürülmesi amaçlanan modellerin ifade edilmesi için uygun bir ortam oluşturmaktadır. BLUE-M modelleri yapısal olarak düğümlerden ve bağlantılardan oluşmaktadır ve anlamsal ağlara benzemektedir. Hem düğümlerin hem de bağlantıların birer etiketi ve kendilerine özgü özellikleri vardır.

Şekil 7.8'de modelleri düzenlemek için kullanılan MEditor (Model Editor anlamında) düzenleyicisi görülmektedir.




Şekil 7.8. BLUE-M Model editörü

Model editörü alışılageldik grafik düzenleme özelliklerini sağlayan, tıklayarak seçme, yer değiştirme, sürükleyip bırakma veya boyunu ayarlama gibi olaylara tepki veren, C++ dilinde yazılmış bir çizge tasarlama editörüdür. Düğümler arasına yerleştirilen bağlantıların, düğümlere kilitlenme ve onlarla birlikte hareket etme özelliği olduğundan, çizgelerin görünümü kolayca istenilen biçimde düzenlenebilmektedir.

Burada düzenlenen modeller, ilişkiler ve özelliklere sahip düğümler ve bağlantılar içermektedir. İlişki ve özellik kavramları birbirlerinden ayrılmıştır. Model gövdesini oluşturan çizgenin haricinde model içine *doğrudan dönüşüm bilgisi* içeren yorumlama kuralları koymak mümkündür. Şekilde *değişken* olarak etiketlenmiş olan eleman bu biçimde çalışmaktadır. Bu yorumlayıcıları tüm model tiplerine yerleştirmek mümkündür. Bu yorumlama kuralları ile birlikte her bir model kendi içsel yapısını da kullanarak kendisine gönderilen mesajlara cevap vermekte ve böylece yorumlayıcı rolünü oynamaktadır. Bir modelin tamamen kurallardan oluşmasını sağlayarak pratikte bir dönüştürücü olarak kullanmak mümkün olduğu gibi, bu kurallara hiç yer vermeyerek kendi içsel yorumuyla sınırlı bir biçimde kullanmak da mümkündür.

Model editörü,BLUE-M modellerini bir hipermedya olarak yorumlar ve bağlantılar üzerinden proje üzerinde dolaşmaya izin verir. Ayrıca en basit yapıda olanlar dışındaki model elemanlarını tıklayarak iç detaylarını ayrı birer model olarak izlemek ve manipüle etmek mümkündür. Bunun dışında çalıştırılabilir modelleri editör içinden ateşlemek ve sonucu izlemek mümkün olabilmektedir.

Desenler, mega model ve arayüz modelleri de temel model editötü kullanılarak yaratılabilir. Desenler fiziksel yapı açısından modelleri tanımlamaktadırlar. Bazı düğüm ve bağlantıları değişken olan modellerdir. Ayrıca bir modelin desen rolü oynaması isteniyorsa, kendine ait arayüz modelinin içinde veya projenin mega-modelinin içinde en az bir pragmatik nokta (dil) açısından desen olarak tanımlanmalıdır. Desenler dönüşümlerde okuma ve yazma işlemlerinde kullanılırlar. Ayrıca bazı kontrol yapılarının içinde yer alırlar. BLUE-M ortamında desen oluşturmak için özel bir düzenleyici bulunmamaktadır. Temel model düzenleyicisi kullanılarak oluşturulabilirler. Düğümlere işaretçiler yerine değişkenler yerleştirerek fiziksel olarak düzenlenirler.

*Mega-modeller* herhangi bir model veya dönüşüm şeması için tanımlanabilen ve bu şemanın işlevini projenin bütününe (ve diğer şemalara) göre tanımlayan özel bir model türüdür. Modele özgü bir tane oluşturulup tanımladığı modele iliştirildiğinde *arayüz modeli* adını almaktadır. Yapısal olarak yine temel

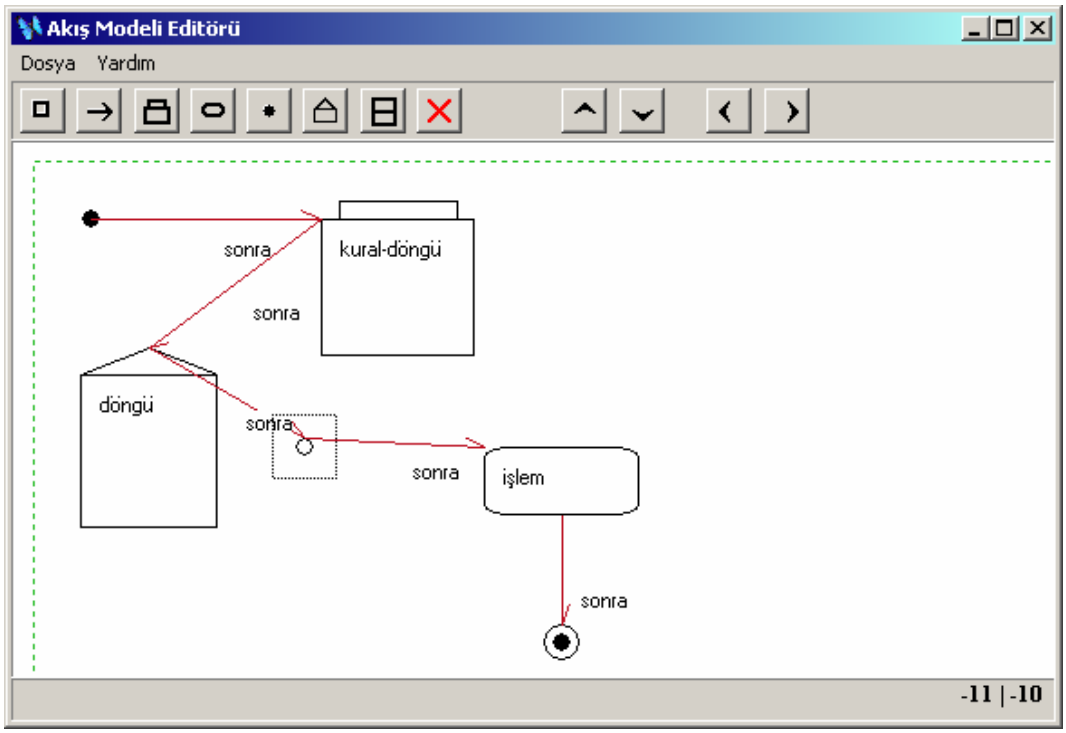
modelden bir farkı yoktur ancak arayüz modeli tanımladığı modelin ilişiğinde tutulan bir model olması ile ayrıcalıklı bir konumdadır. Bir arayüz modeli sadece bir modele ait olabilir. Sadece mega-model olarak adlandırılan biçimi ise modellere eklenmiş olmayıp tüm proje için bir veya birkaç tane yaratılarak, projenin genel bir görünümünü vermeyi amaçlayan mega-modellerdir. BLUE-M ortamı, projedeki tüm mega-ilişkileri devşirerek tek bir büyük mega-model'e dönüştürür. Kendi iç işleyişinde dinamik olarak ürettiği bu mega-modeli kullanır. Bu açıdan arayüz modellerinin belli bir modele bitişik yapıda olmaları yorumlanmalarında herhangi bir yerelliğe yol açmaz. Bu uygulama sadece düzenin sağlanması ve gereken bilgilere hızlıca ulaşılması içindir.

Modeller, arayüzler üzerinden hem kendilerine başvurma hakkı verilen başka modellere bağlanırlar, hem de kendi kaynaklarını başka modellere sunarlar. Üretilen her model, proje çapında başvuru kaynağı olarak işlev gören bir veya birkaç modele eklemlendirilir ve bu başvuru modellerinin içinde bir dil öğesi olarak işlev görür. Benimsenen model yapısı sayesinde farklı tipteki dil öğeleri, aynı isim uzayını paylaşmak dışında herhangi bir yapılandırmaya gerek duyulmadan bir araya gelebilmekte ve ortak bir dil oluşturabilmektedirler. Model yapısı 7.3.1'de detaylı olarak anlatılmıştır. Modeller arası ilişkiler bağlamında arayüz modelleri ise altıncı bölümde detaylı olarak anlatılmıştır.

### 7.3.3 Akış modeli editörü

Akış modeli, temel modelden yapısal ve anlamsal olarak farklı bir model tipidir. Şekilde BLUE-M Akış Modeli Editörü görülmektedir. Model editörüne benzeyen bir pencere olduğu halde ortamın davranış biçimi farklıdır (Şekil 7.9).

Akış modelinde tanımlanan yapının elemanları arasındaki sıra önemlidir ve bu sırayı gösteren tek kancalı kırmızı oklar mevcuttur. Akış modeli dönüşümün akışını tanımlamakta kullanılan modeldir. Flowchart ile state diagram'ın bir birleşimi olarak düşünülebilir. Akış içinde birbiri ardınca gelen elemanların arasında *durum* (*state*) noktaları tanımlamak mümkündür.




Şekil 7.9 BLUE-M Akış modeli editörü

Akış modeli içinde olayların akışı düz bir çizgi üzerinde gerçekleşir. Dallanmalar ve döngüler, model elemanları içinde bırakılmışlardır. Örneğin bir döngü içindeki akışın kodlanması gerektiğinde o döngüye ait eleman sağ tıklanarak yeni bir akış editörü penceresi açılmaktadır.

Akış editörü özel bir ilişki türü olarak belirlenmiş olan sıra ilişkisini (şekilde *sonra* olarak etiketlenmiş) otomatik olarak kurmaktadır. Eklenen her eleman bir önceki son elemandan bir sonra gelecek biçimde sıraya yerleşmektedir. İlk sıradaki bir *başlama düğümü* ve son sıradaki *bitiş düğümü* yine sistemin özel düğümleridir ve otomatik olarak yaratılmaktadırlar. Editör içindeyken elemanları ekleyip silmenin yanında, sıralarını değiştirmek de mümkündür. Sürükleyip bırakılan her model elemanı ekranda kendi üstündeki elemanlardan sonra, altındaki elemanlardan ise önce gelecek biçimde yeniden sıralanmaktadır.

Bu model düzenleyicisi sadece çalıştırılabilir kodu değil elemanları sıralı olan herhangi bir modeli düzenleme görevine sahiptir. Modellerin hipermedya yapısında olmasının bir sonucu olarak araştırılan ve içeriği görüntülenmek istenen sıralı listeler, komutlar, çok parametreli ilişkiler ve bunlardan türetilen her türlü veri yapısı akış editörünün çalıştırılmasına neden olmaktadır.

# 8. SONUÇLAR

Bu tez çalışmasında model güdümlü mühendislik için semiyotik kavramlarına dayanan bir modelleme ve model dönüşümü yaklaşımı önerilmiştir. Bu yaklaşımın bütünleşik bir platformda somutlaştırılması için BLUE-M *Basic Linguistic Unification Environment for Modeling* (Modeller için Basit Dilsel Birleştirme Ortamı) geliştirilmiştir. Bu çalışmayı diğer yaklaşımlardan ayıran en temel özellik, birleştirme ilkesinin tavizsiz bir uygulamasına dayanıyor olması ve model kavramının son derece kapsayıcı bir yorumundan hareket etmesidir. Bu yaklaşımın, model tabanlı yazılım geliştirmenin kendi başına bir paradigma olması açısından önemli olduğu düşünülmektedir.

Bu çalışma "Her şey modeldir" biçiminde ifade edilmiş olan saf bir model güdümlü yazılım geliştirme paradigmasına duyulan ihtiyaca cevap vermeyi dileyen ve bu amaçla semiyotik biliminin (işaret-bilim) doğal dilleri açıklamak için kullandığı kavramlara başvuran bir bakış açısını savunmaktadır. Her ne kadar bir yazılım geliştirme paradigmasının yerleşmesi, güçlü ve zayıf yanlarının belirlenmesi için uzunca bir süre denenmesi gerekse de; model kavramının genel bir yorumu çerçevesinde alan bilgisinden başlayarak kullanılan teknolojilere kadar tüm yazılım geliştirme sürecinin entegre edilmesi ile, otomasyon için mükemmel bir çalışma ortamı sağlanacağı ve böylece verimlilik, güvenilirlik ve değişikliklere uyum konusunda ciddi iyileşmeler olacağı açıktır.

Klasik bakış açısında dönüşümler her zaman modellere yukarıdan bakan yapılar oldukları halde bu çalışmada, model kavramının programsal gerçekleştiriminde, dönüşüm kavramı doğrudan kullanılmıştır. BLUE-M yaklaşımında modeller, çalıştırılabilen, sorgulanabilen ve etkileşime girebilen nesnelerdir. Dolayısıyla modellerin içinde roller üstlenen dönüşümler de vardır. Bu model anlayışının sağladığı esneklik, birleştirme ilkesinin gerçekleştirilmesi öngörüsünün bir parçasıdır.

Bu bakış açısının bir uzantısı olarak model kavramının da son derece jenerik bir yorumu benimsenmiştir. Modellerin ilişkisel bilgi veren çizge (graph) yapısında nesneler olmaları bir zorunluluk değildir. Model, temelde sorgulara yanıt verebilen ve bir başka nesneyi temsil etme yeteneği olan bir yapıdır. Bu temsil edişin somut biçiminin belirlenmesinde semiyotik bilimi devreye girmiştir. Modellerin sadece benzeşim kurularak oluşturulmadığı, bunun yanında referans yapmak, genellemek, mantıksal önermelerde bulunmak gibi soyutlama çeşitlerinin

bir arada kullanıldığı bir model anlayışı benimsenmiştir. Bunlar semiyotik bilimindeki *paradigma, syntagmata* (veya tam karşılığı olmamakla birlikte *syntax*) ve *semantik* kavramlarının karşılığıdır ve bağlam ve yorumlama ilişkilerini kontrol eden *pragmatik (pragmatics)* kavramıyla birlikte, önermiş olduğumuz model güdümlü paradigma anlayışının dört temel perspektifini oluşturmaktadır.

BLUE-M modelleri birden fazla modelleme rolünü yerine getirmektedirler, bu roller kaynağını semiyotik biliminden alan dört temel perspektife ait ilişkiler yoluyla tanımlanırlar. Bu tercih, her bir modelin, bir veya daha çok linguistik (dilsel) rol oynadığı anlayışından kaynaklanmaktadır. Böylece modeller, otomasyon destekli biçimsel (formal) bir ortamda, diller oluşturacak biçimde birleşmektedirler. Bu ortam içinde yazılım geliştirme süreci, yazılımla ilgili alanı, değişik soyutluk düzeylerinde birkaç defa tarayan farklı dillerin geliştirilmesi biçiminde ilerlemektedir. Örneğin tipik bir bilişim sisteminde, iş mantığı, donanım, iletişim, veri tabanı ve kullanıcı arayüzü için farlı diller oluşacaktır. Bu anlayış, katmanlı mimari gibi yaygın mimari yapıların, geliştirme paradigması tarafından daha iyi desteklenmesi anlamına gelmektedir.

Önerilen yaklaşımda pragmatik perspektif'in bir gereği olarak tüm modellerin birer dil uygulaması (yorumlayıcı) olarak görülmesi mümkündür. Bu yorumlayıcılar soyutluk yelpazesinin bir ucunda gerçek modelleme dillerini içerebilmekte, diğer ucunda ise anlık olarak oluşturulan, mesaja tepki verme özellikleri ile kodun kendisi ve veri içeren nesneleri kapsamaktadır. Arada farklı soyutlama biçimleriyle birçok dil bulunabilmektedir. Bu yapı, klasik bakış açısı olan *platform bağımsız model* ve *platforma özgü model* içeren ikili model anlayışından çok daha zengin bir soyutlama zenginliğini temsil etmektedir. Ayrıca katmanlandırılmış veya hiyerarşik olarak düzenlenmiş diller biçiminde, yeniden kullanıma çok daha açık bir proje yapısı ortaya çıkmaktadır. Dil kavramı tüm ortam için kapsayıcı bir rol oynamaktadır.

Önerilen paradigmanın işlerliği, arayüz modelinin yapısı ve işlevi konusuna odaklı örnekler yardımıyla gösterilmeye çalışılmıştır. Bu kapsamda bir tasarım deseninin, bir kayıt yapısının ve bir dönüşümün sisteme entegrasyonu anlatılmıştır.

BLUE-M projesi kapsamında fonksiyonel bir programlama diline benzer sözdizim ile kodlanan ve tasarım geliştirme ve dönüştürme ortamının çekirdeğini

oluşturan BLUE-KOD dili yaratılmıştır. BLUE-KOD ortamı çalışır durumdadır ve grafik arayüz ile entegre biçimde çalışabilmektedir. Prosedür yazma ve çağırma imkanları vardır. Parametre alan ve değer döndüren prosedürler, istenildiğinde kendi yerel bağlamında istenildiğinde dinamik bağlama (*dynamic binding*) yoluyla global bağlamda çalıştırılabilmektedir. Dile güçlü kontrol yapıları kazandırılmış ve basit programları yazıp çalıştırabilecek seviyede bir anlatım gücüne ulaşılmıştır. Komut ara yüzü esneklik ve kontrol edilebilirlik bakımından istenen düzeye gelmiştir. Programlama dillerindeki komutlara benzer kontrol yapılarıyla birlikte MOLA benzeri bir akış kontrolü sağlanmıştır. Ayrıca hem BLUE-M modellerini hem de dönüşümlerini görsel bir ortamda yaratmak ve düzenlemek amacıyla iki görsel editör yaratılmıştır. Görsel modellerin içeriği ile BLUE-KOD isim uzayları birbiriyle özdeş hale getirilmiştir ve böylece ortamın entegrasyonu sağlanmıştır. Geliştirme ortamındaki model tasarımında, model kavramının esnek biçimde uygulanmış olması, işaretçi sınıfının (PlastPointer) güçlü tasarlanmış olması (Fiziksel veya sembolik işaret olabiliyor) ve tüm dil elemanlarının yorumlayıcı ve çalıştırılabilir kod olarak görülebilmesi, kodlama sürecinde savunulan paradigmaya bağlı kalınmasının bir göstergesidir.

Tez çalışmalarına ilişkin olarak iki ulusal (Egesoy (Sıkıcı) ve Topaloğlu, 2007; Egesoy ve Topaloğlu, 2009) ve iki uluslar arası yayın yapılmıştır (Egesoy (Sıkıcı), 2005; Egesoy and Topaloğlu , 2009).

Bu tez kapsamında yürütülen çalışmanın, saf bir model güdümlü geliştirme paradigmasına duyulan gereksinime karşı, semiyotik bilimi kavramları kullanılarak verilen bir yanıt olması amaçlanmıştır. İstenilen etkiyi sağlayan tek çözümün burada savunulan (semiyotik) çözüm olması beklenemez. Bu çalışmanın en azından saf bir model paradigmasının avantajlarına dikkat çekmesi ve benzer çalışmalara vesile olması umulmaktadır.

Bu çalışma açısından gelişmeye açık olan konular aslında ancak kodlamanın tamamlanmasından sonra gerçek projelerde denenmesi aşamasında ortaya çıkacaktır. Tasarım üzerinde bazı değişiklikler ihtimal dahilindedir. Bunun dışında paradigmanın oturmasından sonra gündeme gelebilecek konular arasından bu aşamada bilebileceklerimiz de vardır. Örneğin alan modellemenin gereksinimlerinin araştırılması ve karşılanmaya çalışılması gündeme getirilebilir. Alan modeli girdisi olarak önceden kodlanmış ontolojik bilgilerin kullanılması gibi konular, gelişmeye açık geniş alanlar sunmaktadır. Projenin denenmesi ve

rafine edilmesi süreci ile birlikte farklı alanlardaki modelleme performansının değerlendirilmesi gerekecektir.

Göreli uzak bir hedef olarak da olsa *tamamen model güdümlü bir programlama* anlayışının uygulanabilirliği değerlendirilmelidir. Birleştirme ilkesinin kaçınılmaz bir sonucu olarak tasarım ile kod arasındaki engel doğallığını kaybedecek ve hata ayıklama ve değişikliklere uyum sağlama gibi görevlerde sağlayacağı avantajlar nedeniyle, tek parça bir model güdümlü sürecin kodlamayı da kapsayacak biçimde gerçekleştirilmesi gündeme gelebilecektir. Bu çalışmada, birleştirme ilkesine gösterilen özen ile bu konuda da gelişmeye açık bir anlayışı desteklemek hedeflenmiştir.

# KAYNAKLAR

**Aachen Üniversitesi,** "PROGRES: A Graph Grammar Programming Environment", Aachen Üniversitesi'nin resmi PROGRES Web sitesi, http://www.se.rwth-aachen.de/tikiwiki/tiki-index.php%3Fpage=Research%3A+Progres.html (Erişim Tarihi: 27 Ocak 2010)

**Agrawal, A.**, 2004, Model Based Software Engineering: Graph Grammars and Graph Transformations, Area Paper, Vanderbilt University, EECS, 48p (unpublished).

**Alvarez, J., Evans, A. and Sammut, P.**, 2001, MML and the Metamodel Architecture, Workshop on Language Descriptions, Tools and Applications, (LTDA), http://www.cs.york.ac.uk/puml/mmf/SammutWTUML.pdf, (Erişim Tarihi: 15 Ocak 2010).

**Atkinson, C. and Kühne, T.**, 2001, The Essence of Multilevel Metamodeling, UML 2001: 4th International Conference, in LNCS, Springer, 2185:19-33.

**Atkinson, C. and Kühne, T.**, 2002, The Role of Metamodeling in MDA, In J. Bezivin, & R. France (Eds.). Proceedings of UML'2002 Workshop in Software Model Engineering (WiSME 2002).

**Baresi, L. and Heikel, R**., 2002, Tutorial Introduction to Graph Transformation: A Software Engineering Perspective, Proceedings of the 1st International Conference on Graph Transformation, LNCS, 2505: 402-429.

**Bezivin, J.**, 2005, On the Unification Power of Models, Software and Systems Modeling, 4:171-188.

**Chandler, D.**, 1994, Semiotics for Beginers, http://www.aber.ac.uk/media/Documents/S4B/semiotic.html, (Erişim Tarihi: 6 Eylül 2010)

**Christoph, A.**, 2004, Describing Horizontal Model Transformations with Graph Rewriting Rules, Proceedings of MDFA-2004,: 93-107.

**Egesoy (Sıkıcı) A.**, 2005, Modeling from a Semiotic Perspective, ACM International Conference Proceeding Series; Vol. 214, Proceedings of the 2005 symposia on Metainformatics, (MIS'05), Esbjerg, Denmark, 214:14-26.

**Egesoy (Sıkıcı) A. ve Topaloğlu, N.Y.,** 2007, Yazılım Geliştirmede Model Sınıflandırma, 3. Ulusal Yazılım Mühendisliği Sempozyumu (UYMS'07) Bildiri Kitabı.

**Egesoy, A. and Topaloğlu, N.Y.**, 2009, A Bottom-up Model for Computational Semiotics, Proceedings of the 11th International Conference on Informatics and Semiotics in Organizations, 2009, (ICISO 2009) Aussino Academic Publishing House, pp 26-32.

**EgeSoy, A. ve Topaloğlu, N.Y.,** 2009, Yazılım Geliştirmede Anlam Sorunları ve Model Güdümlü Yaklaşım, IV. Ulusal Yazılım Mühendisliği Sempozyumu ve Sergisi (UYMS09) Bildiri Kitabı, (332):143-148.

**Favre, J.M., and Nguyen, T.**, 2004, Towards a megamodel to model software evolution through transformations, SETRA Workshop, Elsevier ENCTS, http://www-adele.imag.fr/Les.Publications/intConferences/SETRAa2004Fa

v.pdf, (Erişim Tarihi: 30 Ocak 2010).

**Favre, J.M.**, 2004, Towards a Basic Theory to Model Model Driven Engineering, Proc. of the 3rd UML Workshop in Software Model Engineering (WISME2004), LNCS, 3297:43-51.

**Gamma, E., Helm, R., Johnson, R., and Vlissides, J.,** 1994, Design Patterns: Elements of Reusable Object-Oriented Software, 403p.

**Kalnins, A., Barzdins, J., and Celms, E.**, 2004, MOLA Language: Methodology Sketch. - Proceedings of EWMDA-2, Canterbury, England, pp.194-203.

**Königs, A.**, 2005, Model Transformation with Triple Graph Grammars, in Model Transformations in Practice Satellite Workshop of MODELS 2005, http://sosym.dcs.kcl.ac.uk/events/mtip05/submissions/konigs__model_transformation_with_triple_graph_grammars.pdf, (Erişim Tarihi: 27 Ocak 2010)

**Königs, A., and Schürr, A.**, 2006, Tool Integration with Triple Graph Grammars- a Survey, Electronic Notes on Theoretical Computer Science,:113-150.

**Kurtev, I.**, **Bezivin, J., and Akşit, M.,** 2002, Technological Spaces: an Initial Approach, Proceedings of the International Symposium on Distributed Objects and Applications, Irvine, USA.

**Liu, K.**, 2000, Semiotics in Information Systems Engineering, Cambridge University Pres, Cambridge, 218p.

**McNelie, A.**, 2003, MDA: The Vision with the Hole?, http://www.metamaxim.com/download/documents/MDAv1.pdf, (Erişim Tarihi: 7 Eylül 2010)

**Mens, T., Czarnecki, K., and Van Gorp, P.,** 2006, A Taxonomy of Model Transformations, Proceedings of the International Workshop on Graph and Model Transformation, (GraMoT-2005), 152:125-142.

**Minsky, M.L.**, 1968, Matter, Mind and Models. Semantic Information Processing, ed Marvin Minsky, MIT Pres.

**Moflon Org.**, MOFLON resmi Web sitesi, Darmstadt Üniversitesi, http://www.moflon.org/, (2009) (Erişim Tarihi: 27 Ocak 2010).

**Peirce, C.S.**, 1931, Collected Writings ed Charles Hartshorne, Paul Weiss & Arthur W. Burks, Harvard University Pres, Cambridge, MA.

**Schürr, A.**, 1991, PROGRES: A VHL-language based on graph grammars. Proc. of the 4th Int. Workshop on Graph-Grammars and Their Application to Computer Science, LNCS, Springer-Verlag, 532:641–659.

**Schürr, A.**, 1994a, PROGRES: A Visual Language and environment for Programming with Graph Rewriting Systems, Technical Report AIB 94-11, RWTH Aachen.

**Schürr, A.**, 1994b, Specification of Graph Translators with Triple Graph Grammars. In G. Tinhofer, editor, WG'94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science, LNCS, Springer Verlag, 903:151–163.

**Seidewitz, E.**, 2003, What Models Mean, IEEE Software, 20(5):26-32.

**Selic, B.**, 2003, The Pragmatics of Model-Driven Development, IEEE Software, 20(5):19-21.

**Sendall, S. and Kozaczynski, W.**, 2003, Model Transformation: the Heart and Soul of Model-Driven Software Development, IEEE Software, Special Issue on Model Driven Software Development, 20(5):42-45.

**Sowa J. F.,** 2000, Ontology Metadata and Semiotics, Ganter & G. W. Mineau, eds., Conceptual Structures: Logical, Linguistic, and Computational Issues, Lecture Notes in AI , Springer-Verlag, Berlin, 1867:55-81.

**Thomas, D.**, 2004, MDA: Revenge of the Modelers or UML Utopia, IEEE Software, 21(3):15-17.

# EKLER

Ek 1 BLUE-KOD Başvuru Kılavuzu

**Ek 1**

# BLUE-KOD Başvuru Kılavuzu

## Tamsayı sabitleri

Alışageldik şekilde yazılır

**Ör:** `12  -43  0`

## Metin Sabitleri

Çift tırnak içinde yazılır. (Metin sabitleri içinde şimdilik çift tırnak kullanılamamaktadır.)

**Ör:** `"Bu bir metin "`

## Mantıksal sabitler

Mantıksal sabitler `evet` (mantıksal doğru) ve `hayır` (mantıksal yanlış) sözcükleriyle ifade edilir.

**Ör:** `(ve;evet;hayır)`

## Semboller

Programlama dillerindeki değişken isimlerinin karşılığıdır. Bir harf ile başlaması zorunludur. Harf-sayı dizisi ile devam eder. Uzunluk sınırı yoktur. Alt çizgi '_' kullanılmamaktadır. Büyük-küçük harf ayrımı vardır. Türkçe karakterler desteklenmektedir.

**Ör:** `myVar1`

# Fonksiyonlar

Lisp sözdizimi benzeri biçimde yazılır. Parantezin ilk elemanı fonksiyon adını belirten semboldür. Elemanlar arasında noktalı virgül vardır.

**Ör:** `(bağla;A;B)`

Fonksiyonlar birbiri içine gömülebilir.

**Ör:** `(bağla;A;(topla;12;B))`

# Fonksiyon Kataloğu

## Aritmetik

### (bağla;Arg1;Arg2)

Basit değer bağlama (atama) fonksiyonudur. İki argüman alır. Arg1 argümanı bir semboldür; Arg2 argümanı ise hesaplanabilen herhangi bir değerdir. Bağlama sırasında herhangi bir tip kontrolü yapılmaz. Bağladığı değeri aynı zamanda döndürür. (C' deki atama benzeri). Bağlam içinde var olmayan bir sembole atama yaparak onun yaratılması sağlanabilir. Var olan bir sembole atama yaparak ise eski değerinin yok olmasına yol açmak mümkündür.

**Ör:** `(bağla;myVar;12)`

### (topla;Arg1;Arg2;...)

Tamsayı değerlerini toplayan fonksiyondur. En az bir argüman almaktadır. Argümanlar hesaplanarak her birinden bir tamsayı değeri elde edilir ve bu değerler toplanır. Sonuç döndürülür.

### (çıkar;Arg1;Arg2)

Bir tamsayı değerlerini diğerinden çıkaran fonksiyondur. İki argüman almaktadır. Argümanlar hesaplanarak her birinden bir tamsayı değeri elde edilir. Arg1 değerinden Arg2 değeri çıkarılır. Sonuç tamsayı döndürülür.

### (çarp;Arg1;Arg2;...)

Tamsayı değerlerini çarpan fonksiyondur. En az bir argüman almaktadır. Argümanlar hesaplanarak her birinden bir tamsayı değeri elde edilir ve bu değerler toplanır. Sonuç döndürülür.

### (böl;Arg1;Arg2)

Bir tamsayı değerlerini diğerine bölen fonksiyondur. İki argüman almaktadır. Argümanlar hesaplanarak her birinden bir tamsayı değeri elde edilir. Arg1 değeri Arg2 değerine kalansız bölünür. Sonuç tamsayı olarak döndürülür.

### (arttır;tRef)

Bir tamsayı referansının değerini bir arttırır. Aynı zamanda tamsayının yeni değerini döndürür.

**Ör:**

```
(bağla;i;4). //i'ye 4 atadık
(arttır;i).  //i 5 oldu
(bağla;j;(arttır;i)).  //i ve j 6 oldu
```

### (azalt;tRef)

Bir tamsayı referansının değerini bir azaltır. Aynı zamanda tamsayının yeni değerini döndürür.

**Ör:**

```
(bağla;i;4). //i'ye 4 atadık
(azalt;i).  //i 3 oldu
(bağla;j;(azalt;i)).  //i ve j 2 oldu
```

### (büyük;Arg1;Arg2)

Tamsayı karşılaştırması yapar. Arg1, Arg2'den büyükse "evet" (mantıksal doğru) döndürür. Aksi halde "hayır" (mantıksal yanlış) döndürür.

### (küçük;Arg1;Arg2)

Tamsayı karşılaştırması yapar. Arg1, Arg2'den küçükse "evet" (mantıksal doğru) döndürür. Aksi halde "hayır" (mantıksal yanlış) döndürür.

### (eşit;Arg1;Arg2)

Tamsayı karşılaştırması yapar. Arg1, Arg2'ye eşitse "evet" (mantıksal doğru) döndürür. Aksi halde "hayır" (mantıksal yanlış) döndürür.

### (büyükeşit;Arg1;Arg2)

Tamsayı karşılaştırması yapar. Arg1, Arg2'den büyükse veya Arg2'ye eşitse "evet" (mantıksal doğru) döndürür. Aksi halde "hayır" (mantıksal yanlış) döndürür.

### (küçükeşit;Arg1;Arg2)

Tamsayı karşılaştırması yapar. Arg1, Arg2'den küçükse veya Arg2'ye eşitse "evet" (mantıksal doğru) döndürür. Aksi halde "hayır" (mantıksal yanlış) döndürür.

**Ör:** `(küçükeşit;23;(topla;15;5))`

**(ve;M1;M2;..)**

Mantıksal ve işlemidir. Sınırsız sayıda mantıksal argüman alır. Tümünün değeri evet ise evet döndürür.

**(veya;M1;M2;..)**

Mantıksal veya işlemidir. Sınırsız sayıda mantıksal argüman alır. En az birinin değeri evet ise evet döndürür.

**(değil;M)**

Mantıksal değilleme işlemidir. Bir tane mantıksal argüman alır. Argüman değeri evet ise hayır, hayır ise evet döndürür.

**Ör:** Mantıksal fonksiyonları kullanarak karmaşık mantıksal ifadeler yazmak mümkündür.

```
(değil;(veya;(küçük;34;25);(eşit;33;(topla;30;3))))
```

Bu ifadenin C'deki yazılışı: `!((34<25)||(33==(30+3)))`

# Kontrol Komutları

### (işlet;Arg)

İşletme ve hesaplama fonksiyonudur. Bir argüman almaktadır. Önce Arg değerini hesaplar, sonra bu değerin içeriğini tekrar hesaplar. (işletir). "koru" fonksiyonunu etkisiz hale getirir ve içeriğini çalıştırır.

### (yişlet;Arg)

İşlet fonksiyonunun yerel bağlamda çalışan versiyonudur. Kodun işletilmesi için var olan aktivasyon kaydını kullanmaz. Yeni bir yerel aktivasyon kaydı (ve dolayısıyla yerel bir bağlam) oluşturarak işletme işlemini buna gerçekleştirir.

### (koru;Arg)

Lisp'teki üstten virgül (quote) operatörünün karşılığıdır. Arg'ı hesaplamaz. Bunun yerine Arg görevinde olan komut ağacını kopyalar..

**Ör:** `(koru;(topla;23;dd))` Toplama işleminin sonucunu değil kimliğini döndürür. Bu işlev sayesinde kod bir değişkende saklanabilir.

**Ör:** `(bağla;kod1;(koru;(topla;23;dd)))`

daha sonra bu kod parçası değişken ismi aracılığı ile işlet fonksiyonuna gönderilerek çalıştırılabilir.

```
(işlet;kod1)
```

### (koru;Arg1;Arg2;..)

Koru komutu birden fazla argüman alacak şekilde de kullanılabilir. Bu durumda argümanlar yine hesaplanmaz fakat bir sıralı listenin içine konularak döndürülür. Sıralı listeler kod içerebilirler ve çalıştırıldıklarında içerdikleri kodu sırasıyla çalıştırırlar. ayrıca *döndür* komutu ile değer döndürme özellikleri vardır. *Döndür* komutu kullanılmazsa *yok* değeri döndürülür.

**Ör:**

```
(bağla;kod1;(koru;(bağla;res;(topla;x;2));(döndür;res))
).
(işlet;kod1).
```

**Ör:**

```
(işlet;(koru;(yaz;"Merhaba dünya");
              (yaz;"Merhaba evren"))
        )
```

### (eğer;M;K1;K2)

Programlama dillerindeki if deyiminin karşılığı. M mantıksal ifadesi hesaplanır, doğru ise K1, yanlış ise K2 çalıştırılır. K1 ve K2 fonksiyon sabiti olabileceği gibi fonksiyon içeren değişken de olabilir.

**Ör:** `(eğer;(büyük;k;ortalama);(arttır;b);(arttır;k))`

### (döngü;ilkle;şart;arttır;kod)

Çeşitli döngülerin gerçekleştirilmesini sağlar. C'deki for döngüsünün benzeridir. Her bir turun döndürdüğü değeri bir sıralı liste içine atarak tümünü döndürür.

İlkle, şart ve arttır bölümlerinin ürettiği değerler göz ardı edilmekte ve yok edilmektedir.

Dört argümanın tümü veya bir bölümü boş bırakılabilir. Şart argümanını boş bırakmak sonsuz döngüye yol açar. Gerçek sonsuz döngüler tur sayısının sınırlandırılması ile engellenmiştir. (Standart sınır 100000) Sınır aşıldığında hata mesajı yayınlanır.

**Ör:** Aşağıdaki kod birden 99 a kadar olan sayıları ekrana yazmaktadır.

```
(döngü;(bağla;i;1);(küçük;i;100);(bağla;i;(topla;i;1));(göst
er;i))
```

# Giriş/Çıkış Komutları

### (yaz; değer[;hedef])

Verilen önceden tanımlanmış referans hedefe verilen değeri yazar. *hedef* üzerine yazılabilme potansiyeline sahip bir nesne olmak zorundadır. Dosya, ekran ve benzer çıktı ortamlarını temsil eden nesneler buna örnektir. *değer* herhangi bir hesaplanabilen değer olabilir. yazma işleminin nasıl bir sonuca yol açacağı tamamen üzerine yazılan hedef nesnenin davranışına bağlıdır. İkinci argüman verilmezse değer ekrana yazılır. Yazma işlemi başarıya ulaşırsa mantıksal doğru (evet) değeri döndürülür. Komutun kendi yapısından kaynaklanmayan bir nedenle yazma işleminin başarısızlığa uğraması durumunda mantıksal yanlış (hayır) döndürülür. Diğer hatalarda (örneğin yazılacak değerin hesaplanamaması gibi) hata mesajı verilir.

**Ör:**

```
(yaz;”Merhaba dünya!”).
(yaz;”Merhaba dünya!”;(konsol)).
```

### (konsol)

Konsol nesnesine bir referans döndürür.

### (mdosya;Dosya;Yol;Uzantı)

Metin dosyası nesnesi yaratan bir fonksiyondur. Dosya ismini içeren metin tipinde en az bir en çok üç argüman alır. Birinci argüman dosya ismi, ikinci argüman dosyanın dizin yolu (directory path) dir. Üçüncü argüman ise dosyanın uzantısıdır. Dosya yolunu yazarken ters kesme işaretinin C metinsel sabitlerinde olduğu üzere çift yazılması gerektiği unutulmamalıdır. İkinci ve üçüncü argümanlar opsiyoneldir. İkinci argüman atlanırsa varsayılan dizin, geçerli dizin altındaki “data” ismindeki bir dizin kabul edilir. Üçüncü argüman atlanırsa varsayılan uzantı “.dön” olmaktadır. Uzantılar başlarındaki nokta ile birlikte yazılırlar. **mdosya** fonksiyonu yarattığı nesneyi döndürür. Bu metin dosyası nesnesi herhangi bir değer gibi değişkenlere atanabilir. Daha sonra bu değişken aracılığı ile dosyaya ulaşmak ve girdi/çıktı işlemleri gerçekleştirmek mümkündür.

**Ör:** `(bağla;myFile;(mdosya(“dene1”)))`

**Ör:** `(bağla;text;(oku;(mdosya;"dene";;”.txt”)))`

### (oku;Arg)

Okuma fonksiyonudur. Okunabilir bir nesneden bir değer okuyup bunu döndürür. Öncelikle Arg hesaplanır. Elde edilen nesneden okuma işlemi gerçekleştirilir. Okunan değer döndürülür.

**Ör:**

```
(oku;myFile)
(oku;(konsol))
```

### (derle;Str)

Script derleme fonksiyonudur. Str metin ifadesini çalıştırılabilir koda dönüştürür. Döndürdüğü tip bir *sıra (sequence)* içinde dizili "*entry*" ağaçlarıdır. Her bir ağaç bir komut satırını karşılar. Her bir *entry* de bir fonksiyonu karşılar (fonksiyonel programlama anlamında). döndürülen bu yapı *işlet* komutu ile çalıştırılabilir. Derleme sırasında veya çalıştırma sırasında oluşan hataların bildirimi ilgili komutların, komut satırından girilmesinde olduğu gibidir. Script içinde komutlar biribiri ardınca yazılır ve fonksiyonlar iç içe yazılarak öbekler oluşturulabilir. Bunlar arasında ayraç karakteri olarak nokta "." kullanılır. Çalıştırma sırasındaki değişken isimlerinin yorumlanmasında dynamic binding uygulanır.

**Ör:** Bir dönüşüm dosyası dosya sisteminden yüklenir, derlenir ve çalıştırılır.

```
(bağla;text;(oku;(mdosya;"dene";".\\data\\";".dön"))).
(bağla;code;(derle;text)).
(işlet;code).
```

### (şema)

Boş bir şema yaratır.

**Ör:**

```
(bağla;şemam;(şema)). //şemam adında yeni şema yarat
```

### (ilişki;Kaynak;Tip;Hedef)

Bir ilişki yaratır. Girdi olarak ilişkinin kaynağı, tipi ve hedefi verilir. Yaratılan ilişki nesnesi döndürülür. Kaynak, tip ve hedefin her üçü de yaratılan işaretçilerdir. İlişkiler ekle komutu ile şemalara eklenebilirler. İşaretçilerin yerel metinlere işaret ettiği durumda şemaları Web sitelerine dönüştüren çweb nesnesi, ilişkileri de navigasyon bağlantılarına dönüştürür.

### (ekle;ŞemaAdı;İlişki)

Adı verilen şemaya verilen ilişkiyi ekler. *ŞamaAdı* referans değişkenidir, *İlişki* ise değer değişkenidir. Adı verilen şemanın aktif bağlamda tanımlı olması gerekir. İlişki'nin ise hesaplanabilir bir değer olması gerekir.

**(göster; Arg)**

Adı verilen önceden tanımlanmış bir değişkenin değerini ekranda gösterir.

**(metin;[ilkdeğer])**

Boş bir metin nesnesi yaratır.

**Ör:**

```
(bağla;M;(metin)). //M adında yeni metin yarat
```

**(yapıştır;MetinR; MetinD)**

Metin değişkeninin sonuna verilen metini ekler. MetinR girdisi metin referansıdır. Önceden tanımlanmış olan referans aktif bağlamda bulunur. MetinD girdisi ise herhangi bir metinsel değerdir. Bu değer ile MetinR 'nin sonuna ekleme (append) işlemi yapılır.

**(yama;Bütün;Pozisyon;Parça)**

Metin değişkeninin verilen bir noktasına verilen bir metin parçasını ekler.

**Bütün :** Hedef alınan metin referansı.

**Pozisyon:** Ekleme işleminin kaçıncı harften itibaren yapılacağını gösteren tamsayı

**Parça:** Eklenecek olan metin değeri

Ekleme işlemi, eklenen noktadan sonraki metin parçasını, eklenen yamanın boyu kadar sağa kaydırır (insert işlemi). Mantıksal doğru değeri döndürür.

**(mbul;Bütün;Parça)**

Bir metin parçasını daha büyük bir metin içinde bulur. Alt metnin bulunduğu pozisyonu tamsayı olarak döndürür. Bulamama durumunda sıfır döndürür.

**(çweb)**

Şemalardan basit bir web sitesi yaratabilen bir çevirmen nesne yaratır. çweb, çizge-web çeviricisi anlamına gelir. Bir şema değerini, bu nesne üzerine yazarak karşılık gelen bir Web sitesinin sayfaları (daha sonra, önceden tanımlı bir dizinde yaratılmak üzere) saklanabilir. çweb nesneleri aynı zamanda çalıştırılabilir nesnelerdir ve yazma işleminden sonra işlet komutu ile çalıştırıldıklarında gerçek dosyalar diske yazılır.

**Ör:**

```
(bağla;web;(çweb)). //Web yaratıcısı
(bağla;şem;(şema)).
(ekle;şem;"Kurt").
(ekle;şem;"Havuç").
(ekle;şem;"Geyik").
(yaz;web;şem).
(işlet;web). //dosyaları yarat
```

Aynı işlem, tüm çizge yerine tek tek ilişkileri yazarak da yapılabilir.

**Ör:**

```
(bağla;ww;(çweb)).   //Web üreticisini yarat
(yaz;ww;(ilişki;"Kurt";"Tüketir";"Geyik")).
(yaz;ww;(ilişki;"Geyik";"Tüketir";"Çimen")).
(işlet;ww)
```

## (eleman;id1;id2;..idN)  (under constr)

Kayıt ve dizi yapısı üzerinde dolaşmayı sağlar.id1'den başlayarak bir dizi isimlendirme elemanı istenilen verinin indeksi olmaktadır. İsimlendirme elemanları tamsayı olduklarında (dizi indeksi gibi) herhangi bir liste nesnesinin (şema, küme vs.) istenilen sıradaki elemanına, bir metinsel sembol olduğunda ise (kayıt hanesi gibi) herhangi bir isim uzayı nesnesinin verilen isimdeki elemanına erişim sağlar.

# Çevrim Komutları

**(tamsayı;metinDeğeri)**

Metin değerini okuyarak tamsayı değeri yaratır. Okuma sırasında öncelikle metin içindeki boşluk karakterlerini atlar ve sonrasında rakam olmayan bir karaktere rastlayana kadar sayıyı okur. sayıdan sonra gelen farklı karakterler okuma işlemini başarısızlığa uğratmaz.

**Ör:**

```
(bağla;x;(tamsayı;(girdi;"Bir sayı giriniz:"))).
```

# Alt-Yordam Komutları

### (parametreler;s1;s2;..sN)

Bir dönüşüm aktivasyonundaki parametre değerlerine daha kolay erişmek amacıyla referans sembolleri oluşturur. Bu fonksiyon çalıştırıldığı zaman yerel değişken isimleri yaratılır ve birinci parametre s1, ikinci parametre s2 adını alır.Böylece dönüşüm script'i içinden parametrelere erişmek daha kolay olur ve okunurluk da artar.

**Ör:** Ortalama bulan bir script

```
(parametreler;p1;p2).
(döndür;(böl;(topla;p1;p2);2)).
```

### (döndür; Arg)

Alt-yordamdan değer döndürmede kullanılır.

**Ör:**

```
//Script formatı
(topla;2;3).
(topla;4;5). //Buraya açıklama yazılabilir
(bağla;ee;(topla;6;3)).
(döndür;(çarp;ee;3)).
```

# ÖZGEÇMİŞ

**Ahmet (SIKICI) EGESOY**

**Adres:** Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, 35100 Bornova İzmir.

**Adres (Ev):** 204/24 sok. No:2 (Blok A) Kat 7, D.26, Atatürk Mah. Buca, İzmir..

**Tel:** 3434000-5334 **Tel (Cep):** 533 2461681

**E-Mail:** ahmet.s@ege.edu.tr

**Doğum Yeri/Tarihi :** İzmir / 09 .04. 1971. **Medeni Hal :** Evli

**Yabancı Dil:** İngilizce (KPDS-A) **Askerlik Durumu :** Tamamlandı

## Eğitim(Yeniden eskiye)

**Doktora** Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, İzmir. [2004-Devam ediyor] (Tahmini bitiş tarihi Şubat-2010 )

**Doktora Tezi:** Model Tabanlı Yazılım Geliştirme İçin Semiyotik Bir Model Dönüşüm Dili Tasarımı ve Gerçekleştirimi

**Danışman:** Prof. Dr. N. Yasemin Topaloğlu

**2000 – Yüksek Lisans:** Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, İzmir. (1998-2000)

**Yüksek Lisans Tezi:** Bulanık Uzman Sistem Kabuğu, ( Fuzzy Expert System Shell ), Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü.

**Danışman:** Prof. Dr. Emrah Orhun

**1994 – Lisans:** Orta Doğu Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü, Ankara. (1990-1994)

**Lisans Bitirme Projesi:** Automated Theorem Prover (Otomatik Teorem İspatlayıcısı), Orta Doğu Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü, Danışman:

**Danışman:** Dr. İsmail H. Toroslu

**1988 -** İzmir Özel Türk Lisesi. Sosyal Bilimler Dil ve Edebiyat Kolu (1981-1988 İngilizce hazırlık sınıfı ile birlikte)

## Uluslararası Yayınlar (Yeniden eskiye)

**[1]** Ahmet Egesoy, N.Yasemin Topaloğlu**, A Bottom-up Model for Computational Semiotics,**

Proceedings of the 11th International Conference on Informatics and Semiotics in Organizations, 2009, (ICISO 2009) Aussino Academic Publishing House, 2009, p 26-32.

ISBN:978-0-9806057-2-3

**[2]** Ahmet Sıkıcı, **Modeling from a Semiotic Perspective,** ACM International Conference Proceeding Series; Vol. 214, Proceedings of the 2005 symposia on Metainformatics, (MIS'05), Esbjerg, Denmark, Article No. 14, 2005, ISBN:978-1-59593-719-3.

**[3]**Ahmet Sıkıcı, N.Yasemin Topaloğlu, **Strategies for Hypermedia Design Modeling,** Metainformatics Symposium, Graz, Austria, Revised Papers Series: Lecture Notes in Computer Science, Vol. 3002 Hicks, David L. (Ed.) 2004, ISBN: 3-540-22010-0.

**[4]** Ahmet Sıkıcı, N.Yasemin Topaloğlu, **A Pattern Based Approach to Web Design Formalization,** Turkish Journal of Electrical Engineering & Computer Sciences, Vol. 12 Issue 2, 2004, p 117-126, ISSN: 1300-0632.

**[5]** Ahmet Sıkıcı, N.Yasemin Topaloğlu, **Towards Software Design Automation with Patterns**, Informatica (Slovenya), Vol.25, No. 3 (2001) 309-317.

**[6]** Ahmet Sıkıcı, N.Yasemin Topaloğlu, **Towards Building with Design Patterns,** Proceedings of The Fifteenth International Symposium on Computer and Information Sciences (ISCIS XV), Ekim, 11-13, 2000, İstanbul, Türkiye. Akademi Yayıncılık, Ankara, Türkiye. ISBN:975-6885-06-8.

**[7]** Ahmet Sıkıcı, N.Yasemin Topaloğlu, **A Knowledge Based Approach to Design with Patterns,** Knowledge-Based Software Engineering : Proceedings of the Forth Joint Conference on Knowledge-Based Software Engineering (JCKBSE 2000), Eylül, 12-14, Brno, Çek Cumhuriyeti. IOS Press, Amsterdam, Hollanda, 2000, ISBN:1-58603-060-4

**[8]** Ahmet Sıkıcı, Emrah Orhun, **A Fuzzy Shell for Intelligent Tutoring,** Proceedings of the International Conference on Information Technology Based Higher Education and Training (ITHET 2000), Temmuz 3-5, 2000, İstanbul, Türkiye. Boğazici Üniversitesi Basımevi, İstanbul, Türkiye, ISBN:975-518-143-1.

**[9]** Ahmet Sıkıcı, N.Yasemin Topaloğlu, **Design Patterns as Building Blocks in Reusable Software Design,** Proceedings of The Fourteenth International Symposium on Computer and Information Sciences (ISCIS XIV), Ekim,18-20,1999 Kuşadası, Türkiye. Ege Üniversitesi Basımevi, İzmir, Türkiye. ISBN:975-483-422-9.

## Ulusal Yayınlar (Yeniden eskiye)

**(I)** Ahmet (Sıkıcı) EgeSoy, N.Yasemin Topaloğlu, **Yazılım Geliştirmede Anlam Sorunları ve Model Güdümlü Yaklaşım**, IV. Ulusal Yazılım Mühendisliği Sempozyumu ve Sergisi Bildiri Kitabı (UYMS09) Ekim 2009.

**(II)** Ahmet Sıkıcı, N.Yasemin Topaloğlu, **Yazılım Geliştirmede Model Sınıflandırma**, 3. Ulusal Yazılım Mühendisliği Sempozyumu (UYMS'07) Bildiri Kitabı, Eylül 2007.

**(III)** Ahmet Sıkıcı, N.Yasemin Topaloğlu, **Web Tasarım Mantığının Desen Tabanlı Biçimselleştirilmesi,** 1. Ulusal Yazılım Mühendisliği Sempozyumu (UYMS'03) Bildiri Kitabı, Ekim 2003, ISBN: 975-395-741-6.

**(IV)** Ahmet Sıkıcı, N.Yasemin Topaloğlu, **Web Tasarımına Desen Tabanlı Bir Yaklaşım**, Bilgi Teknolojileri Kongre ve Fuarı (Bilgitek 2003) Bildiri Kitabı, 1-4 Mayıs 2003.

## Yayınların Tarandığı Bilinen İndeksler

[1] ISTP

[2] DBLP, Google Akademik, ACM Portal

[3] SCI, DBLP

[4] Sciencestage, Perfspot, CiteSeer

[5] DBLP, Microsoft Academic Search, io-Port.net

[7] SCI

## Çalışma Deneyimi

**Aralık 2002 – Halen:** Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü

**Görev:** Araştırma Görevlisi

**Şubat 1997 – Mart 1998:** EgePorcan Bilgisayar Sistemleri San. ve Tic. A.Ş Izmir. (Univera)

**Görev:** Yazılım Mühendisi (Tasarım ve kodlama)

**Teknik İçerik:** Janus uyumlu el terminalleri üzerinde çalışan, gezici dağıtım elemanlarına yönelik sıcak-satış ve ön-satış uygulaması, ve toplanan bilgilerin

NETSIS ve LOGO benzeri ofis programlarına aktarılabilmesine olanak sağlayan PC arayüzü.

**Aralık 1995 - Aralık 1996:** Milli Savunma Bakanlığı, ANKARA.

**Görev :** Programcı Subay (Askerlik hizmetim)

**Teknik İçerik:** Informix üzerinde veri tabanı geliştirme ve işletme.

## Asistanlığı Yapılan dersler

BIM447 Yazılım Mühendisliği

BIM219 Programlama Dilleri

BIM107 Algoritma ve Programlama I

BIM112 Algoritma ve Programlama II

BIM111 Ayrık Yapılar

BIM203 Elektrik Devreleri

## Diğer Görevler

Eylül 2008 - II. Ulusal Yazılım Mimarisi Konferansı, Yerel Düzenleme Kurulu üyeliği.

Erasmus Programı Bölüm Koordinatör Yardımcılığı (Sürüyor)