ÜST MODELE DAYALI MODEL DÖNÜŞÜMLERİ

Özlem Morkaya[1] Tahir Emre Kalaycı[2]

[1] Rektörlük Bilgi İşlem Daire Başkanlığı, Ege Üniversitesi, İzmir
[2]Bilgisayar Mühendisliği Bölümü, Ege Üniversitesi, İzmir

[1]e-posta: ozlem.morkaya@ege.edu.tr [2]e-posta: tahir.kalayci@ege.edu.tr

Özetçe

Model tabanlı yazılım geliştirme, yazılım mühendisliğinde nesneye yönelik programlamadan daha üst düzeyde soyutlama olanağı sunan bir yaklaşımdır. Bu çalışmada model tabanlı yazılım geliştirmede gerçekleştirilen üst modele dayalı model dönüşümleri incelenmiş ve bu kapsamda model ve üst model kavramları tanıtılarak, model tabanlı yazılım geliştirmedeki önemi açıklanmıştır. Üst modele dayalı model dönüşümlerinin gerçekleştirilmesi için var olan model dönüşüm araçlarından ATL ve GReAT kullanılarak üst model ve dönüşüm kuralları tanımlanmış ve bu araçların sahip olduğu nitelikler ve sunduğu olanaklar karşılaştırılmıştır.

1. Giriş

İlk bilgisayarlar 1 ve 0'lardan oluşan makine dili komutları ile programlanmaktaydı. İlk bilgisayarlarda program geliştirenler için makine dilinden, soyutlama düzeyi daha yüksek ve insanlar için daha anlamlı komutlar sunan birleştirici (assembly) dil ile programlamaya geçmek şüphesiz önemli bir yenilik idi. Birleştirici dilinden günümüze ise programlama dilleri ve platform teknolojilerinde önemli gelişmeler gerçekleştirildi. Bu alanda ortaya çıkan yeniliklerin temel hedefi soyutlama düzeyi daha yüksek programlama ortamları sunmaktır.

Ancak buradaki soyutlama, çalışma alanının ("domain") kendi problem uzayı için değil, çalışma alanı için bilgisayar ortamında geliştirilen yazılımların çözüm uzayı içindir. Yazılım teknolojilerinin her geçen gün uygulama geliştiricilere sundukları soyutlama düzeyinin artmasına paralel olarak bu teknolojilerin platformları daha da karmaşık hale gelmektedir. Bunun yanında uygulama geliştiriciler sürekli bu teknolojilerin hızlı değişimini takip etmek, mevcut yazılımları en güncel teknolojilere uyarlamak ve eski teknolojilerin yeni teknolojiler ile birlikte çalışabilmesini sürdürebilmek durumundadır. Model tabanlı yazılım geliştirme yukarıda bahsedilen bu sorunlara çözüm getirmek için önerilmiş bir yaklaşımdır. Model tabanlı yazılım geliştirmeden beklenen mevcut yazılımların farklı teknolojilere uyarlanmasını mümkün olduğunca insan müdahalesiz gerçekleştirmektir. Bunun yanında uygulama geliştiricilerin kod yazımından daha çok gerçek dünya uygulamalarının çözüm uzayında yoğunlaşmalarını sağlamaktır [1].

Model tabanlı yazılım geliştirmenin en önemli özelliği bir sistemin farklı yönleri ile modellenmesi ve bir modelden diğerine otomatik çevrim için dönüşümlerinin tanımlanmasıdır. Model tanımlamasında gerçekleştirim detayları en sona bırakılarak modellerin daha taşınabilir, yeni teknolojilere daha uyumlu olması ve farklı teknolojiye sahip diğer sistemlerle daha kolay birlikte çalışması sağlanır [2].

Model Tabanlı Mimari ("Model Driven Architecture"-MDA) OMG tarafından modellerin yazılım geliştirme sürecinde kullanılabilmesi için önerilen ve sistemin çalışma belirtimlerini, sistemin kullandığı platformun detaylarından ayırmak amacıyla geliştirilen bir yazılım tasarım yaklaşımıdır. Model Tabanlı Mimari;

- Sistemi platformdan bağımsız tanımlamak
- Platformları tanımlamak
- Sistem için platform seçmek ve
- Sistem tanımlamasını belirli bir platforma dönüştürmek

için gerekli yaklaşımı sağlar ve araçların geliştirilmesi hakkında tanımlamalar içerir [17].

MDA, hesaplama bağımsız model (CIM (Computation Independent Model)), platform bağımsız model (PIM (Platform Independent Model)), ve platform bağımlı (PSM (Platform Specific Model)) modelleri ve bunların arasındaki dönüşümleri tanımlamaktadır. Hesaplama bağımsız model (CIM), sistemi çalışma ortamı içersinde, sistemin ihtiyaçlarını bilgisayar bağımsız şekilde tanımlar. Platform bağımsız model (PIM) ise sistemin fonksiyonlarını gerçekleştirim ayrıntılarından uzak olarak tanımlar. Buradaki en temel hedef Platform bağımsız modelden (PIM) platform bağımlı modelin (PSM) elde edilmesidir. Bu yöntem; fonksiyonelliğin tanımlaması ile hedef platforma ait gerçekleştirim ayrıntıları arasında yüksek seviyeli bağımsızlık sağlar. Bu yazılım mühendislerine platform bağımsız modelden (PIM) platform bağımlı modele (PSM) otomatik dönüşümü gerçekleştirme olanağı sağlar [17] .

Model tabanlı yazılım geliştirmede tanımlanan bir modelin geçerli bir model olup olmadığını doğrulamak için üst modele ihtiyaç vardır. Üst model, bir modelin belirtimini tanımlar. Bunun yanında modelden üst modelin ve üst modelden de modelin elde edilmesi model tabanlı yazılım geliştirme açısından önemlidir. Bu tersinirlik özelliği sayesinde modelleme dili elemanları ile üst modelleme dilinin elemanlarının bire bir eşlenmesi mümkün olmaktadır [3].

Bu çalışmada hedeflenen model dönüşümünün uygulamada ne şekilde gerçekleştiğini somut biçimde göstermek ve bu kapsamda ATL ve GReAT model dönüşüm araçlarını tanıtmak ve bunları karşılaştırmaktır.

ATL MOF/QVT tabanlı bir araç iken, GReAT grafiksel dönüşüme dayanan ve Model Tümleşik Hesaplama (Model Integrated Computing (MIC)) yaklaşımı çerçevesinde geliştirilmiş bir araçtır. MIC yaklaşımı, MDA yaklaşımı kapsamında model dönüşümü gerçekleştirmenin yanında

dönüşüm sonucu üretilen platforma bağlı modelden (PSM) yola çıkarak bir programlama diline ait kod üretimini de gerçekleştirir [10]. Literatürde MIC ve MDA yaklaşımlarını karşılaştıran pek fazla örnek olmadığı için bu çalışma kapsamında MIC yaklaşımının örneği olan GReAT ile MOF/QVT tabanlı MDA yaklaşımının örneği olan ATL 'in karşılaştırılması yapılmıştır.

Bunun yanında literatürde model dönüşüm araçlarını karşılaştırmak için yapılmış çalışmalar genel olarak bu araçları türlerine göre sınıflandırmayı hedefler ve bu araçları ayrıntılı olarak tanıtmaz. Ancak yaptığımız bu çalışmada her iki model dönüşüm aracının model dönüşümü gerçekleştirmede takip ettikleri adımlar ayrıntılı olarak ele alınmış ve okuyucunun bu adımları izleyerek model tanımlama ve dönüşümü aşamalarını somut olarak izlemesine olanak tanınmıştır.

Bu çalışmada üst model kavramı tanıtılacak ve üst modele dayalı model dönüşümlerinin ne şekilde gerçekleştirildiği incelenecektir. Ardından ATL ve GReAT model dönüşüm araçlarının üst model ve model tanımlamak için sunduğu olanaklardan bahsedilecek ve bu araçların model dönüşümlerini gerçekleştirmede nasıl bir yol izlediği üzerinde durulacaktır. Bunun ardından bu model dönüşüm araçlarının farklı özellikleri karşılaştırılacaktır.

2. Üst Model ve Model Dönüşümü

2.1. Üst Model

Bir üst model ("Meta Model") bir modelleme dilinde geçerli olan modelleri tanımlar. Bir modelin söz dizimsel yapısının nasıl olacağını o modelin üst modeli belirler. Modelin söz dizimsel olarak doğrulanması için modelin üst modeline ihtiyaç vardır. Örneğin UML'in üst modeli UML tanımını oluşturur ve geçerli UML modellerinin belirlenmesini sağlar.

Bir modelin üst modeli o modelin modelidir [4]. Soyutlama hiyerarşisi olarak üst model, modelin üstünde yer alır ve modeller bu üst modelin "örnekleri" ("instance of") olur. Ancak model ve üst model arasındaki "örnekleme" ("instance of") ilişkisi sadece üst modelleme dilinin teorisi için geçerlidir. Üst modelleme dili elemanları ile modelleme dili elemanları arasında geçerli değildir. Bu nedenle üst model ve model arasındaki örnekleme ilişkisini nesneye yönelik yaklaşımdaki sınıf ve örneği arasındaki ilişki gibi düşünmemek gerekir [3].

2.2. Model Dönüşümü

Bir veya daha fazla girdi model üzerine bir takım dönüşüm kuralları uygulayıp sonuçta bir veya daha fazla çıktı model üretmeye model dönüşümü ("Model Transformation") denir [5]. Tüm girdi ve çıktı modellerin üst modeli olmalıdır. Ancak bu şekilde girdi ve çıktı modellerin söz dizimsel olarak doğruluğu tespit edilebilir.
Literatürde model dönüşüm türlerini kullanılan tekniklere göre sınıflayan bir çalışma Sendall ve Kozaczynski tarafından [5]'de önerilmiştir. Bu çalışmaya göre üç tür model dönüşüm türü bulunmaktadır. Bunlar;

- Doğrudan Model İşleme (Direct Model Manipulation)
- Ara Gösterim (Intermediate Representation)
- Dönüşüm Dili Desteği (Transformation Language Support)

2.2.1. Doğrudan Model İşleme

Doğrudan model işleme yönteminde kullanılan API'ler Java, VB veya bunlara benzer programlama dilleri olduğu için dönüşümleri yazmada yeni bir dil kullanılmasına gerek yoktur. Ancak programlama dilleri genel amaçlı olduğundan bu bir dönüşüm için gereken üst seviye soyutlamayı kısıtlar. Dönüşümü kodlama çok zaman alır. Bunun yanında kullanılan algoritmaların bakımı oldukça zor olur [5].

2.2.2. Ara Gösterim Yöntemi

Ara gösterim yöntemi ise, XML tabanlı UML modelleri arasında çevrim için geliştirilen bir standart olan XMI'dan modele, modelden XMI'a dönüşüm tekniğini kullanır. XML'e çevrilmiş bir modele, XSLT gibi mevcut XML araçları ile model dönüşümü uygulanabilir. Wagner'ın çalışmasında [6] XSLT yerine XMI.difference adlı bir araç önerilmiştir, ancak XMI.difference XSLT tabanlı yaklaşımlara göre daha az ifade gücüne sahiptir [5] .

2.2.3. Dönüşüm Dili Desteği

Dönüşüm dili desteği ise model dönüşümlerini tanımlamak için çalışma alanına ("domain") özgü bir dil kullanılmasını önerir. Bu kapsamda model tanımlama ve çalıştırmaya yönelik çok sayıda dil mevcuttur. Bu sınıfa giren dönüşüm dilleri genel olarak tanımlayıcı ("declarative"), emirsel ("imperative"), veya her ikisinin birleşimi özellikte olabilir. OCL'in ("Object Constraint Language") katkısı ile tanımlanan UML profilleri bu noktada önemlidir [5].

3.Model Dönüşüm Araçları

Model dönüşüm dillerinde bulunması gereken özellikler Mens ve Gorp [20] tarafından incelenmiştir. Bu çalışmalarında model dönüşümü için bir sınıflandırma sunmaktadırlar. Bu sınıflandırma bir çok amaç için kullanılabilmektedir.

Model dönüşümlerini gerçekleştirmek için çok sayıda araç ve dil bulunmaktadır. Czarnecki ve Helsen [21] var olan farklı yaklaşımları ve araçları karşılaştırmak için bir özellik modeli sunmaktadırlar. Ayrıca yaklaşım ve araçların incelemesini de bu kapsamda gerçekleştirmişlerdir.

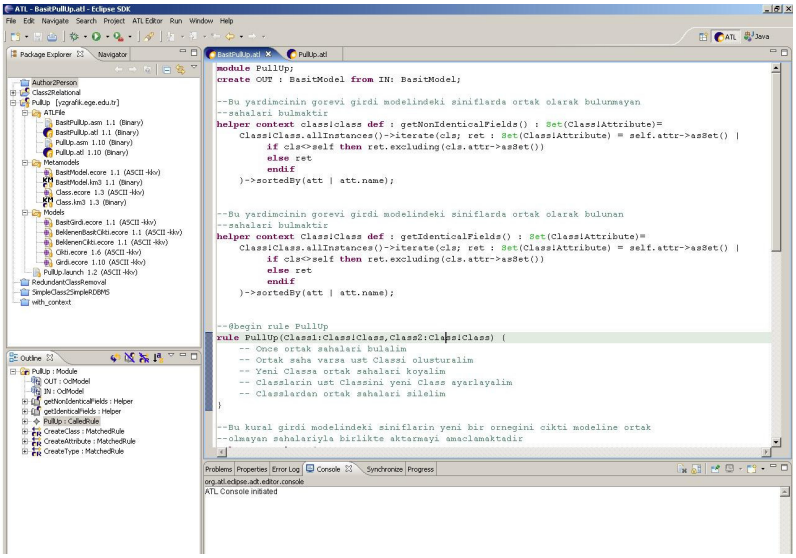
Bu bildiride tanıtılacak olan ATL ve GReAT model dönüşüm araçlarının dönüşüm türü Sendall'ın sınıflandırmasındaki üçüncü tür olan dönüşüm dili desteği grubuna girmektedir. Aşağıda bu dönüşüm araçları tanıtılmıştır.

3.1.ATL

"Object Management Group" tarafından yayınlanan MOF/QVT tabanlı, modelleri dönüştürmek, görüntülemek ve sorgulamak amacıyla alana özgü dil geliştirme isteğine [12] yanıt olarak Nantes üniversitesi ATLAS grubu ve TNI-Valiosys şirketi tarafından önerilen bir model dönüşüm dilidir [13]. Günümüzde ATL projesi Eclipse GMT için dönüşüm araçları sağlamaktadır. Bu araçlar arasında

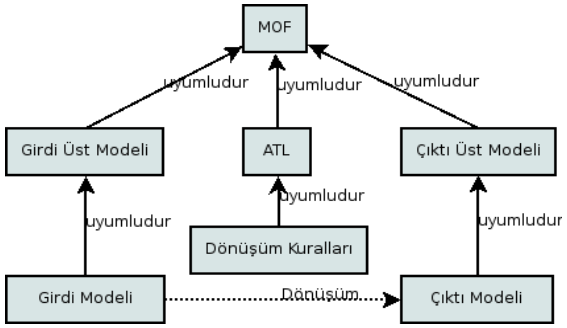
- ATL dönüşümü için gereken kütüphaneler,
- ATL dönüşüm motoru
- Eclipse eklentisi şeklindeki bir ATL geliştirme ortamı

bulunmaktadır.



Şekil 1: ATL Geliştirme Ortamı

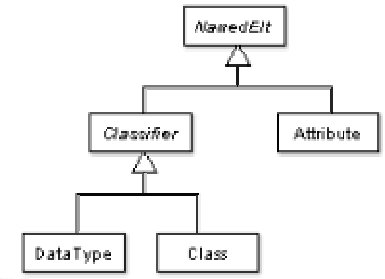
ATL, MDA (Model Driven Architecture) biçiminde model dönüşümleri tanımlamaya dayanmaktadır. Bu dönüşümler açıkça tanımlanmış üst model tanımlarından yararlanılarak oluşturulur. ATL'de her dönüşüm bir modeldir. Şekil 2'de ATL'deki bir dönüşüm ve üst model uyumlulukları incelenebilir [16]. Şekilden de görüleceği gibi ATL dönüşümleri MOF tabanlı dönüşümlerdir. Bu nedenle OMG MDA belirtimlerine uygundur.



Şekil 2: Bir ATL Dönüşümü [16]

3.1.1. ATL üst model hazırlanması

Üst model tanımlarını kolayca gerçekleştirebilmek için ATLAS ekibi tarafından "Kernel Metametamodel (KM3)" adı verilen ve basit metinsel gösterim sağlayan dil geliştirilmiştir. Üst modeller ilk olarak bu dilde hazırlanıp XMI biçimine dönüştürülebilmektedir. Ayrıca üst modelleri UML çizim araçlarında hazırlayıp XMI olarak dışa aktararak kullanmak da mümkündür[15]. Örneğin şekil 3'de KM3 kullanılarak tanımlanmış basit bir sınıf üst modeli incelenebilir.




```
package Class {
    abstract class NamedElt {
        attribute name : String;
    }
    abstract class Classifier extends NamedElt {
    }
    class DataType extends Classifier {
    }
    class Class extends Classifier {
        reference super[*] : Class;
        reference attr[*] ordered container : Attribute oppositeOf owner;
        attribute isAbstract : Boolean;
    }
    class Attribute extends NamedElt {
        attribute multiValued : Boolean;
        reference type : Classifier;
        reference owner : Class oppositeOf attr;
    }
}
package PrimitiveTypes {
    datatype Boolean;
    datatype Integer;
    datatype String;
}
```

Şekil 3: KM3 Kullanılarak Yazılmış Bir Sınıf Üst Modeli

KM3 biçiminde hazırlanan üst modeller "Injection" adı verilen yöntem kullanılarak ATL motoru tarafından kullanılabilecek XMI biçimine dönüştürülmektedir. Üst modelimizi tanımladıktan sonra dönüşümün uygulanacağı girdi modeline üst modele uygun olarak yazmak gerekmektedir. Şekil 4'de üst modelimize uygun olarak sadece iki sınıf içeren basit bir model incelenebilir. Modeller XMI biçiminde hazırlanmaktadır.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0"
        xmlns:xmi="http://www.omg.org/XMI" xmlns="Class">
  <Class name="Author" >
    <attr name="name" type="/2"/>
    <attr name="publisher" type="/2"/>
    <attr name="lastPublishDate" type="/2"/>
  </Class>
  <Class name="Reader">
    <attr name="name" type="/2"/>
    <attr name="lastBook" type="/2"/>
  </Class>
  <DataType name="String"/>
</xmi:XMI>
```

Şekil 4: Girdi modelimiz

3.1.2. ATL dönüşüm dili

ATL dönüşüm dili tanımlayıcı ve emirsel yapılara sahip melez bir dildir. Tanımlama dili OCL 2.0 tabanlıdır. Tanımlayıcı ATL'de dönüşüm kurallarla tanımlanmaktadır. Her kural kaynak modeldeki bir model eleman tipine yönelik tanımlanır ve bu tiplere filtreleme işlemleri de uygulanabilir (Örn. "C" ile başlayan sınıflar). Şekil 5'de tanımlayıcı yapı kullanılarak şekil 3'de tanımlanmış olan üst modele uygun şekil 4'teki girdi modelinden aynı elemanları içeren çıktı modelini üreten dönüşüm kuralları incelenebilir.

```
--Girdi modelindeki sinifin sahalarini
--cikti modeline aktaralim
rule CreateClass {
     from girdi : Class!Class
     to cikti : Class!Class (
          name <- girdi.name,
          attr <- girdi.attr,
          isAbstract <- girdi.isAbstract
     )
}

--Girdi modelindeki sahalari yeni
--modele uygun sekilde aktaralim
rule CreateAttribute {
     from girdi : Class!Attribute
     to cikti : Class!Attribute (
          name <- girdi.name,
          type <- girdi.type,
          owner <- girdi.owner,
          multiValued <- girdi.multiValued
     )
}

--Girdi modelindeki sahalarin tiplerini
--yeni modele uygun sekilde aktaralim
rule CreateType {
     from girdi : Class!DataType
     to cikti : Class!DataType (
          name <- girdi.name
     )
}
```

Şekil 5: Basit bir dönüşümün ATL ile ifade edilmesi

Her elemanın dönüşümü için (sınıf, saha ve saha tipi) ayrı kuralların tanımlanması gerekmektedir. Böylece ilgili elemanın dönüşüm işleminin ne şekilde yapılacağı ATL motoruna bildirilmektedir.

Dönüşüm kuralları modül adı verilen metin tabanlı dosya içerisinde yazılmaktadır. Modül aşağıdaki bölümlerden oluşmaktadır[19]:

- Dönüşüm modülüyle ilişkili bazı özelliklerin tanımlandığı başlık bölümü
- Var olan hazır ATL kütüphanelerinin kullanılmak üzere içselleştirildiği "import" bölümü (Bu bölüm zorunlu değildir)
- Java metotlarına benzetebileceğimiz yardımcı ("helper") yapılarını barındıran bölüm
- Kaynak modellerinden hedef modellerini oluşturulmasını sağlayan kurallar

Başlık bölümü: Başlık bölümü dönüşümün adını ve kaynak ve hedef modelin değişken isimlerini tanımlandığı bölümdür.

```
module PullUp;--Modul ismi
create OUT : BasitModel from IN: BasitModel;
```

Import bölümü: Bu bölümde eğer varsa kullanılacak olan hazır kütüphaneler yazılmaktadır.

Yardımcılar: Yardımcılar dönüşümün herhangi bir yerinden ihtiyaç anında çağrılabilecek metotlar olarak görülebilir. Bir yardımcı isim, kapsam tipi, dönüş değeri, ATL deyimi şeklindeki kod bloku ve isteğe bağlı parametrelerden oluşmaktadır. Örnek olarak aşağıdaki yardımcı incelenebilir.

```
--Bu yardimcinin gorevi stringteki ilk
--karakteri kucuk harfe cevirmektir
```

helper context String **def**: firstToLower() : String = self.substring(1, 1).toLower() + self.substring(2, self.size());

Kurallar: ATL dilinde iki tip kural vardır. Bu tipler ATL'nin tanımlayıcı ve emirsel yapılarına karşılık gelmektedir. Eşlenmiş ("Matched") kurallar ATL tanımlayıcı dönüşümünün çekirdeğini oluşturmaktadır. "rule" anahtar kelimesiyle tanımlanırlar ve tanımlamada belirtilen kaynak modeldeki ilgili tipler için ilgili dönüşümleri gerçekleştirerek modeli dönüştürürler. Zorunlu kaynak ve hedef desen tanımlamasına ek olarak, isteğe bağlı yerel değişkenler ve emirsel blok içerebilirler. Örnek olarak Şekil 5'te verilmiş olan eşlenmiş kurallar incelenebilir. Daha önce tanımlamış olduğumuz üst model ve modellere göre bu kurallar girdi modelindeki her sınıf için çıktı modelinde aynı sınıfı oluşturmaktadır. "CreateClass" eşlenmiş kuralı çıktı modelinde ilgili sınıfı yaratmakta ve ilgili sahalarını oluşturup ayarlamaktadır. Çağrılan ("Called") kurallar ATL'in emirsel yapılarını temsil etmektedir. Çağrılan kurallar yardımcıların özel bir şekli olarak görülebilir. Eşlenmiş kurallarda olduğu gibi "rule" anahtar kelimesiyle tanımlanmaktadır. Bir kaynak model kullanmadıkları için eşlenmiş kurallardan farklı olarak kaynak desenine sahip değillerdir. Hedef model yerel değişkenler, parametreler ve modül özelliklerinden oluşturulmaktadır. Örneğin aşağıdaki kod örneğindeki kural her çağrıldığında parametreler kullanılarak yeni bir "Class" sınıfı oluşturulmaktadır.

```
rule NewClass (na: String, isabstract: Boolean)
{
        to p : Class!Class(
                name <- na
        )
        do {
                p.isAbstract <- isabstract;
        }
}
```

ATL dönüşüm dilinin bir başka önemli yapısı sorgu yapısıdır. Bu yapının amacı kaynak modellerden ilkel değerlerin hesaplanmasını sağlamaktır. Sorguların en çok kullanılma nedeni metinsel çıktı sağlamaktır. Aşağıdaki örnekte yazılmış

olan sorgu incelenebilir(Modeldeki sınıf sayısını result.txt dosyasına yazan sorgu).

```
query ClassNb = Class!Class.allInstances()-
>size().toString().writeTo('result.txt');
```

ATL dilinin tüm yapılarını ve kullanım şekillerini öğrenmek için ATL ana sayfasındaki belgeler ve örnek senaryolara ek olarak daha önceden yapılmış olan dönüşümler incelenmelidir[1].

3.2.GReAT

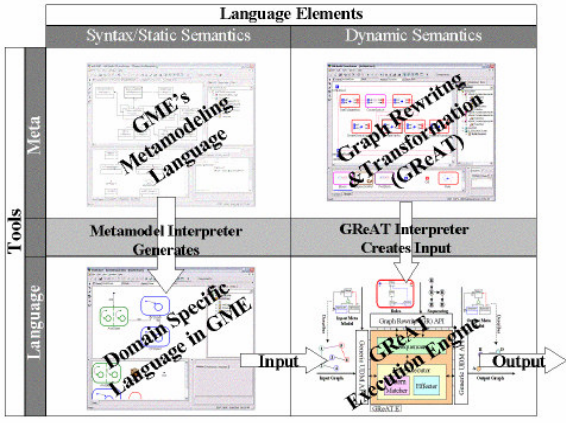
GReAT, Vanderbilt üniversitesi tarafından geliştirilmiş üst modele dayalı model dönüşümü gerçekleştiren bir model dönüşüm aracıdır.

Grafiksel dönüşüm yöntemi ile model dönüşümünü gerçekleştiren GReAT, MIC yaklaşımını destekler. MIC yaklaşımı, MDA yaklaşımından daha eski bir geçmişe sahip olup MDA gibi çıktı olarak platform bağımlı model üretir ve MDA'dan farklı olarak bu modelden yola çıkıp otomatik kod üretimini hedefler [10].

Vanderbilt üniversitesi farklı çalışma alanlarına özgü (domain-specific) model ve üst model tanımlama olanağı sunan ve bu model ve üst modelleri kullanarak model dönüşümünü gerçekleştiren bir çerçeve tanımlanmıştır [7]. Bu çerçeve kapsamında 4 temel bileşen vardır. Bu bileşenlerin ilk ikisi üst model ve model tanımlamada diğerleri ise model dönüşümünde kullanılır. GReAT model dönüşüm aracı olduğu için bu bileşenlerden son ikisini bünyesinde bulundurur. Bu bileşenler aşağıda listelenmiştir:

- GME üst modelleme dili (GME's metamodeling language)
- GME modelleme ortamı (Generic Modeling Environment)
- GReAT model dönüşümü tanımlama dili (Graph Rewriting and Transformation)
- GReAT dönüşüm motoru (Execution Engine (GReAT-E))

Bu bileşenlerin oluşturdukları çalışma alanına özgü geliştirim çerçevesi şekil 6'da gösterilmiştir.



Şekil 6: Çalışma alanına özgü geliştirme çerçevesi [7]

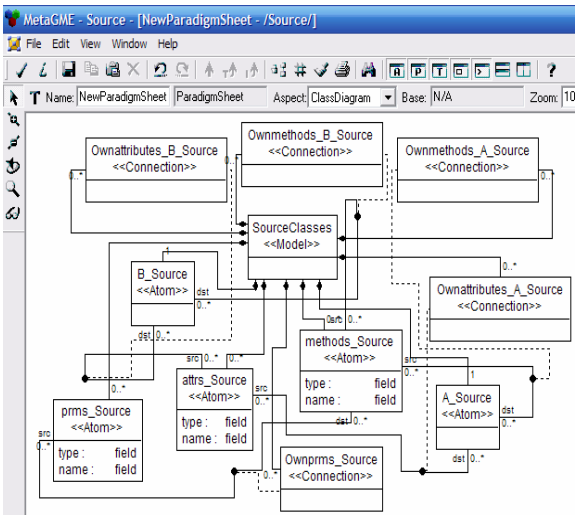
1 http://www.eclipse.org/m2m/atl/

Bu bileşenler ve görevleri aşağıda tartışılmıştır.

3.2.1.GME Üst Modelleme Dili

Grafiksel olarak, alana özgü üst model tanımlama olanağı sunan bileşendir. Burada tanımlanan sisteme paradigma olarak kaydedilen üst modellerden aşağıda anlatılacak olan GME'de çalışma alanına özgü modelleme yapılabilir. Bunun yanında tanımlanan üst modeller GReAT model dönüşüm aracında kullanılır. Üst modellemede söz dizimsel tanımlamalar için UML sınıf diagramları kullanılırken, statik anlamsallık ise OCL ile tanımlanan koşullar aracılığı ile belirtilir.

Üst modelden, model tanımlama ve model dönüşümünde üst modeli kullanmada üst modelin görsel olarak gösterimi için UML'e eklentiler yapmak gerekir. Bu eklentiler genelde önceden tanımlı nesne özellikleridir [8]. Şekil 7'de GME'de tanımlanan bir üst model gösterilmiştir.



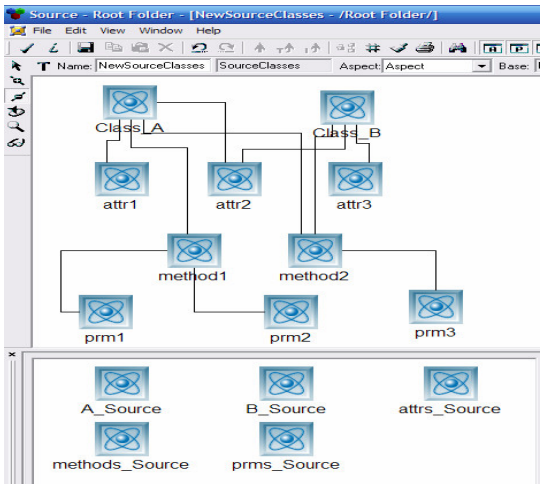
Şekil 7: Bir üst model

Üst modelin kapsadığı UML sınıf diagramları klasörler şeklinde gruplanır. Klasörlerin içinde atom (atom), model (model), görünüm (aspect), referans(references), bağlantı (connection) ve küme (set) olarak adlandırılan kısaca FCO (First class objects) denilen sınıf nesneleri bulunur [8]. Bu kavramlar aşağıda açıklanmıştır.

- Atom: Temel nesnelerdir. Başka nesneleri kapsayamazlar. Önceden tanımlı özellikleri vardır.
- Model: Bileşik nesnelerdir. Başka nesneleri kapsayabilirler. Diğer nesnelerle olan ilişkilerinde mutlaka bir rol üstlenirler.
- Görünüm: Görünüm bir model nesnesinin model tanımlama işleminde nasıl görüneceğini belirlemede kullanılır.
- Referans: Nesneye yönelik programlamada kullanılan işaretçilerin (pointer) görevini üstlenir.
- Bağlantı: Aynı modele ait nesneler arasındaki ilişkiyi gösterir.
- Küme: Bir grup nesne arasında ilişki kurmak gerektiğinde kümeler kullanılır.

3.2.2. GME Alana Özgü Modelleme Ortamı

GME, sisteme çalışma alanına özgü olarak paradigma şeklinde kaydedilen üst modellerden model tanımlama olanağı sunar. Bu ortamda üst modelden model tanımlanması grafiksel olarak gerçekleştirilir. Şekil 8'de, şekil 7'de sunulan üst modelden, model tanımlanması gösterilmiştir.



Şekil 8: Şekil 7'de sunulan üst modelden model tanımlama

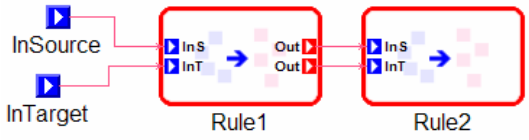
Üst modele ait olan atomlar grafiksel olarak modele eklenir ve bu atomlar arasında üst modelin izin verdiği şekilde bağlantılar kurulur. GME'de sisteme paradigma olarak kaydedilen bir üst modelden model oluşturma olanağı sayesinde üst modele uymayan model tanımlanmasının önüne geçilir. Bu, modelin geçerliliğinin daha modelin oluşturulma aşamasında kontrol edilmesini sağlar.

Burada tanımlanan modeller GReAT model dönüşümü aracında girdi model olarak kullanılır ve bu modeller üzerinde model dönüşümü gerçekleştirilir.

3.2.3. GReAT Model Dönüşümü Tanımlama Dili

Yukarıda da bahsedildiği üzere bu bileşen üst modele dayalı model dönüşümünü tanımlama dilidir. Dönüşüm dili olarak UMT (Universal Model Transformer) adlı bir dil kullanılır. Burada ifade (expression) tüm kural tanımlamaları için taban sınıf görevini görür. İfadeler özelleşerek basit (primitive) kuralları, bileşik (compound) kuralları ve testleri oluşturur. Basit kurallar bir dönüşümü belirten en temel birimdir. Basit kurallara örnek olarak GReAT'teki "Rule" ve "Case" verilebilir. "Test" ise bir dönüşümün koşutsal işlemesini belirtmek için kullanılır. Bir test bir durum (case) içerir durumun sağlanıp sağlanmamasına göre dönüşüm farklı çalışma yolları izler. Bileşik kurallar, dönüşüm dizileri ve grafik karmaşıklığını kontrol gibi nedenlerle kullanılır. Bileşik kurallara örnek olarak GReAT'teki "Block" ve "ForBlock" verilebilir. Bileşik kurallar diğer basit veya bileşik kuralları içerebilir [9].

Bu ifade türleri ve gerektiğinde OCL ile yazılan koruma (guard) ve özellik eşleme (attribute mapping) tanımlamaları kullanılarak model dönüşümü tanımlaması tamamlanır. Sonunda ortaya çıkan model dönüşümü şekil 9'da sunulmuştur.



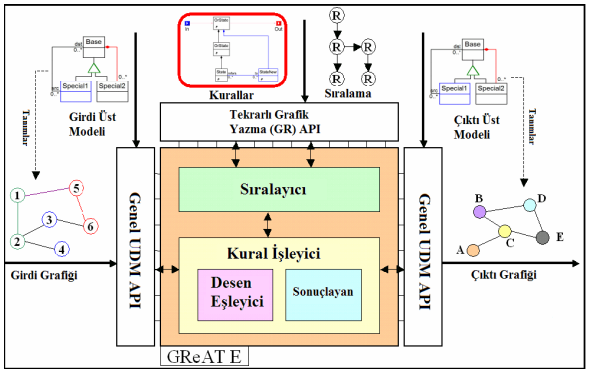
Şekil 9: GReAT model dönüşümü

GReAT model dönüşüm dili 3 bileşenden oluşur. Bu bileşenler aşağıda listelenmiştir. [10]

- Desen belirtim dili (Pattern specification language) : Dönüşüm için kullanılan UML sınıf diagramlarının alt grafiklere ayrılıp bu alt grafiklerin ne tür desenlerden oluştuğunu belirlemede desen belirtim dili kullanılır. Kullanılan algoritmalara göre aynı grafik farklı desenlere ayrıştırılabilir.
- Tekrarlı grafik yazma dönüşüm dili (Graph Rewriting Transformation Language): Grafik dönüşüm dili temel olarak ve grafik grammeri ve dönüşümü GGT (Graph grammars and graph transformations) algoritmalarına dayanır. Dönüşüm esnasında üst modele uyumlu olacak şekilde dönüşüm grafiğinin tekrar düzenlenmesine ihtiyaç duyulabilir [7].
- Akış kontrolü dili (Control flow language) : Burada dönüşüm birimi olan kuralların hangi sırada uygulanağı belirlenir. Akış kontrolü dili sıralama (sequencing), önceden tahminlenememe (non-determinism), hiyerarşi (hierarchy), iç-içe tekrarlama (recursion), test-durum (Test/Case) özelliklerini sağlar.

3.2.4. GReAT-E dönüşüm motoru

GReAT dönüşüm dilinde şekil 10'da mimarisi sunulan GReAT-E (Graph Rewriting and Transformation Execution Engine) model dönüşüm motoru kullanılır.

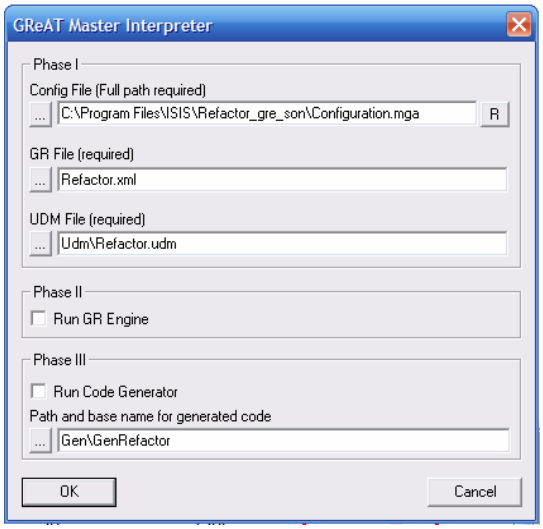


Şekil 10: GReAT-E model dönüşüm motoru [7]

Bu motor bir yorumlayıcı olarak çalışır. Model dönüşümünü veri yapısı şeklinde olan bir program olarak alır ve bu programı girdi grafik üstünde çıktı grafiği üretmek için çalıştırır. Motor API'ler kullanarak her türlü model dönüşümünü gerçekleştirir [7].

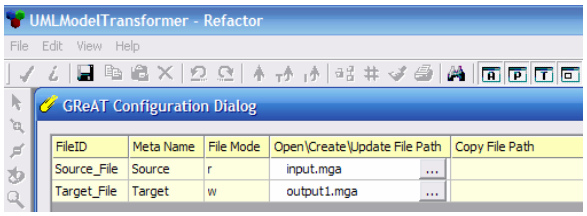
Model dönüşümü tanımlaması yapıldıktan sonra GReAT-E ile bu dönüşüm çalıştırılabilir. Bunun için önce GReAT ortamında ile gösterilen (GReAT Master Interpreter)

yorumlayıcının çalıştırılması gerekir. Bu işlem çalışma alanına özgü UDM dosyaları, konfigürasyon dosyası ve GReAT-E motorunun kullandığı tekrar yazma (rewriting) kurallarını oluşturur [11]. Şekil 11'de yorumlayıcı ekranı gösterilmiştir.



Şekil 11: (GReAT Master Interpreter) Yorumlayıcı ekranı

Bunun ardından GReAT-E'nin çalıştırılması için GReAT'de ile gösterilen (Invoke Engine) dönüşüm motorunun çalıştırılması sonucu girdi modelden çıktı model ve GReAT model dönüşüm aracı aynı zamanda MIC yaklaşımına dayandığı için kod üretimi (Java, C++) gerçekleşir [11]. Bu dönüşüm motoru ekranı şekil 12'de sunulmuştur.



Şekil 12: GReAT-E model dönüşüm motoru ekranı

4. ATL ve GReAT'in Karşılaştırılması

ATL ve GReAT model tabanlı yazılım geliştirmede kullanılan model dönüşüm araçlarıdır. Bu bölümde ATL ve GReAT model dönüşüm dillerinin farklı özellikleri üzerinde durulacaktır.

Model dönüşüm araçlarını tartışırken bu araçların çalışma ortamları, model ve üst model tanımlama için sundukları olanaklar ve model dönüşümü için sahip oldukları özellikler bu araçların kullanım kolaylığını ve verimli kullanılmasını belirleyen önemli karşılaştırma kriterleridir. Bu nedenle bu çalışmada gerçekleştirilen ATL ve GReAT'in karşılaştırılmasında öncelikle gereksinim duydukları çalışma ortamları genel özellikler başlığı altında tartışılacak ardından sahip oldukları model ve üst model tanımlama ortamları ve son olarak da her iki aracın model dönüşümü için sundukları olanaklar tanıtılacaktır.

4.1.Genel Özellikler

ATL, GReAT'e göre daha yeni bir araçtır. Bu nedenle GReAT'e ait döküman sayısı daha fazladır. GME MS/COM (Component Object Model) teknolojisine dayanan bir mimariye sahiptir. ATL dili ise EMF modelleme anaçatısının üzerinde çalışmaktadır [18]. ATL Eclipse Java ortamına entegre edilerek çalışır. GReAT'in ise çalışması için Microsoft VS.NET yazılımına gerek vardır. Her iki araç da ücretsiz olarak kullanılan model dönüşüm araçları olmasına karşın GReAT'in Microsoft VS.NET'e gereksinimi bu noktada ATL'i daha öne çıkarmaktadır.

4.2.Modelleme Ortamı

GReAT'in model tanımlama ortamı grafiksel bir tasarım ortamıdır. Bu kullanıcıya kolay ve zevkli bir çalışma ortamı sağlar. Model tanımlanırken sadece üst modelin izin verdiği bağlantılar model elemanları arasında kurulabilir. Bu sayede daha model tanımlama aşamasında modelin geçerliliği kontrol edilir. ATL'de ise modeller XML döküman dosyalarında tanımlanırken üst model ise bu XML dökümanının ad uzayı olarak belirtilir. Üst modelin ad uzayı olarak belirtilmesi de modelin üst modele uygunluğunun kontrol edilmesini sağlar.

4.3. Üst Model Hazırlanması

GReAT'de UML diagramları benzeri diagramlarla üst model tanımlanır. UML'in kısıtları tanımlamada yeterli gelmediği durumlarda ise OCL ile kısıt tanımlaması gerçekleştirilir. ATL ise üst model tanımlamada grafiksel bir ortam yerine metinsel bir ortam sunar. ATL'de üst model tanımlamak için KM3 dosyaları kullanılır. KM3 olarak tanımlanan üst modeller Ecore veya MOF üst modellerine çevrilebilir. GReAT'de ise tanımlanan üst modeller UML üst modeline çevrilir.

4.4.Dönüşümlerin Tanımlanması

Model dönüşümünde ATL genellikle tanımlayıcı ("declarative") ve biraz da emirsel ("imperative") bir yaklaşım izler. ATL model dönüşümünü ifade etmek için OCL ifadelerini kullanır. ATL, MOF tabanlı model dönüşümü gerçekleştirir. GReAT ise grafik dönüşümüne odaklanmıştır ve grafik grammeri ve dönüşümü GGT (Graph grammars and graph transformations) algoritmalarını model dönüşümünde kullanır. GReAT'de model dönüşümünde grafiksel olarak ifade edemediği kısıtları tanımlamak için OCL 'i kullanır.

GReAT'de dönüşüm tanımlamak için en az bir tane yukarıda açıklanan "block" veya "forblock" tipinde bileşik kural tanımlanmalıdır. Bu bileşik kuralın içine diğer basit veya bileşik kurallar eklenebilir. Basit kurallar ise bileşik kuralların aksine başka kuralları içeremezler. Dönüşüm tanımlanırken bu kuralların hangi sıra ile işleneceği ve her kuralın hangi girdileri alacağı ve hangi çıktıları üreteceği grafiksel olarak belirtilir. ATL'de ise kurallar "atl" uzantılı bir dosyada metinsel olarak tanımlanır. Aynı ATL dosyasında çok sayıda kural tanımı bulunabilir.

5.Sonuç

Model tabanlı yazılım geliştirme, yazılım teknolojilerinin hızla ilerlemesi sonucu mevcut yazılımların yeni teknolojilere uyumu, platform bağımlılığı gibi sorunlara çözüm olarak

önerilmiştir. Model tabanlı yazılım geliştirme uygulama için kod yazımından ziyade uygulamanın çalışma alanının modellenmesine yoğunlaşır.

Model, üst model tanımlama ve model dönüşümleri model tabanlı yazılım geliştirmenin en önemli bileşenidir. Bu çalışmada kapsamında üst modele dayalı model dönüşümlerini gerçekleştiren ATL ve GReAT model dönüşüm araçları kısaca tanıtılmıştır.

Bu çalışmada ATL ve GReAT ortamlarının sunduğu model, üst model ve model dönüşümü tanımlama olanakları incelenmiştir. Bu araçların kullanılması ile ne şekilde model, üst model ve model dönüşümü tanımlaması yapılacağı adım adım anlatılmıştır. Bu sayede, kullanıcıların model tabanlı yazılım geliştirmeyi ve onun aşamalarını kavraması kolaylaşmıştır. Bunun yanında kullanıcıların model dönüşümü ile üretilen çıktı modelleri somut şekilde görmesi de konunun kavranması açısından önemi büyüktür.

Literatürde MDA konusunda yapılan çalışmalar ya ayrıntılı olarak tek bir araç üzerinde yoğunlaşır ya da genel özellikleri ile model dönüşüm araçlarından bahsederek onları sınıflandırır.

Bu çalışmada ise literatürde yapılandan farklı olarak iki ayrı model dönüşüm aracı ayrıntılı biçimde ele alınıp karşılaştırılmıştır. Bu sayede kullanıcılar iki farklı model dönüşüm aracını inceleme olanağı bulmuş ve karşılaştırmalı olarak model dönüşümü aşamalarını görme fırsatı yakalamıştır. Bunun doğal sonucu olarak da kullanıcılar model dönüşümünün aşamaları hakkında daha pekişmiş bilgiye sahip olma imkanı bulmuşlardır.

Model tabanlı yazılım geliştirmenin, hem sağladığı soyutlama artışı hem de üretkenlik artışı ile yakın gelecekte yazılım mühendisliğinde önemli bir araştırma alanı olacağı ve yazılım geliştirmede daha yoğun olarak yer alacağı düşünülmektedir. Bu çalışmada bu alandaki mevcut literatürün ve var olan geliştirme araçlarının kısa bir tanıtımı amaçlanmıştır.

6.Kaynakça

[1] Schmidt, D. C., ``Model-Driven Engineering", *IEEE Computer*, 39(2):25-31, 2006.

[2] Fuentes, L. ve Vallecillo. A., "An Introduction to UML Profiles", *UPGRADE, The European Journal for the Informatics Professional*, 5 (2): 5-13, 2004.

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

[4] Atkinson, C. ve Kühne, T., "Model-Driven Development: A Metamodeling Foundation", *IEEE Software,* 20(5):36-41, 2003.

[5] Sendall, S. ve Kozaczynski, W., "Model Transformation – the Heart and Soul of Model-Driven Software Development", *IEEE Software*, 20(5):42-45, 2003.

[6] Wagner; A., "A Pragmatic Approach to Rule-Based Transformations within UML using XMI.difference", *WITUML: Workshop on Integration and Transformation of UML models (held at ETAPS 2001)*, 2001.

[7] Agrawal, A., Karsai, G., and Ledeczi, A., "An End-to-End Domain-Driven Software Development Framework", *Conference on Object Oriented Programming Systems Languages and Applications*, 2003.

[8] Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle and P. Volgyesi, The Generic Modeling Environment, Vanderbilt University, ISIS (WISP'2001, May, 2001 in Budapest, Hungary), IEEE 2001

[9] Agrawal, Z. Kalmar,G. Karsai, F. Shi, A. Vizhanyo, "GReAT User Manual", ISIS, Vanderbilt University, 2003.

[10] Agrawal, G. Karsai, F. Shi, "Graph Transformations on Domain-Specific Models", GReAT Technical Report, ISIS, Vanderbilt University

[11] GReAT Tutorial, GReAT 1.5.0 Release, 11/07/05, Available at : http://repo.isis.vanderbilt.edu/downloads/,

[12] Object Management Group: OMG/RFP/QVT MOF 2.0 Query/Views/Transformations RFP. October 2002, Available at : http://www.omg.org/docs/ad/02-04-10.pdf

[13] Bezivin, J., Dupé, G., Jouault, F., Pitette, G. ve Rougui, J.E., "First experiments with the ATL modeltransformation language: Transforming XSLT into Xquery", *OOPSLA Workshop on Generative Techniques in the context of MDA,* .2003.

[14] Gerber, A., Lawley, M., Raymond, K., Steel, J. ve Wood, A., "Transformation: The Missing Link of MDA", *ICGT*, 2002

[15] Bezivin, J., Jouault, F., Rosenthal, P., Valduriez, P., "The AMMA platform support for modeling in the large and modeling in the small", *Lina Research Report No:04.09*. 2005

[16] Bezivin J., Jouault F., Touzet D. , "An introduction to the ATLAS Model Management Architecture". *Lina Research Report No:05.01*. 2005

[17] MDA Guide Version 1.0.1, Erişim: http://www.omg.org/cgi-bin/doc?omg/03-06-01

[18] Bézivin, J., Brunette, C., Chevrel, R., Jouault, F. ve Kurtev I., "Bridging the Generic Modeling Environment (GME) and the Eclipse Modeling Framework (EMF)", *OOPSLA Workshop on Best Practices for Model Driven Software Development*, 2005.

[19] ATLAS Group: ATL User Manual, ATL User Manual, version 0.7. 2006. Erişim: http://www.eclipse.org/m2m/atl/doc/ATL_User_Manual%5Bv0.7%5D.pdf

[20] Mens, T. ve Van Gorp, P., "A Taxonomy of Model Transformation", *Proceedings of the International Workshop on Graph and Model Transformation*, 2006.

[21] Czarnecki, K. ve Helsen, S., "Classification of Model Transformation Approaches", *OOPSLA Workshop on Generative Techniques in the Context of MDA*, 2003.