Uçuş Güvertesi Gösterge Sistemlerinin Modellenmesi

Cevahir Turgut[1] Enver Veli Atabek[2]

[1] Enformatik Enstitüsü, Orta Doğu Teknik Üniversitesi, Ankara
[2]Elektrik – Elektronik Mühendisliği Bölümü, Orta Doğu Teknik Üniversitesi, Ankara

[1]cevahir.turgut@gmail.com [2]20393177@mail.baskent.edu.tr

Özetçe

Bu bildiri, "Uçuş Güvertesi Gösterge Sistemleri (Flight Deck Display Systems) - UGGS" için geliştirilen model güdümlü yazılım geliştirme (MGYG) yöntemini geliştirme aşamasında kullanılan tekniklerle ve araçlarla beraber açıklamaktadır. Geliştirilen yöntem kapsamında UGGS'nin alan analizi yapılarak, soyut sözdizimi çıkarılarak, alan kısıtları için iyi tanımlanmış kurallarla, UGGS üst-modeli (*meta-model*) oluşturuldu ve örnek somut söz dizimi verildi. UGGS için EBNF ile gramer oluşturuldu, UGGS için UML 2* profili çıkartıldı, UGGS üst-modeli GMF (Grafiksek Modelleme Çerçevesi) modeline dönüştürüldü ve UGGS modelinden kod üretildi. Çalışmadan elde edilen bilgi birikimi ve tecrübeler kapsamında MGYG yönteminin mevcut durumu artıları ve eksileriyle birlikte verildi. Ayrıca bu bildiri kapsamında geliştirilen yöntemden yola çıkarak MGYG'nin güvenlik kritik yazılımlar için uygulanabilirliği tartışılmaktadır.

1. Giriş

Hava araçları günümüzde cam kokpit sistemleri ile donatılmaktadır. Cam kokpit sistemlerde pilotların hava aracıyla etkileşimleri geçmişte olduğu gibi analog yerine sayısal gösterge sistemleri ile sağlanmaktadır. UGGS pilotlar ile hava araçları (helikopter, uçak gibi) arasındaki temel etkilemişi sağlayan ana sistemlerdir. Etkileşimler cam kokpit sistemleri üzerinden gerçekleşmektedir. Cam kokpit sistemlerinde pilotların hava aracını kontrol edebilecekleri sistemlerde mevcuttur.

Bu çalışma kapsamında uçuş güverte gösterge sistemleri için geliştirilen model güdümlü yazılım geliştirme yöntemi verilmektedir. Bu kapsamda iki temel motivasyondan ilki UGGS'nin bileşenleri diyagramlar ve grafikler ile canlandırmaya uygundur. Bir diğeriyse UGGS için yazılım geliştirme aşamasında modelden kod üretmek maliyet-etkin bir yöntemdir.

Görsel UGGS bileşenleri üzerinden yazılım geliştirmek, bir yazılım geliştirme sorunu olan kompleksliği azaltacağından UGGS için daha üretken, geliştirmesi bakımı daha kolay yazılım geliştirme ortamı sağlayacaktır. Böylelikle yazılım geliştirme ve bakım maliyetleri azalacaktır.

Avyonik ürünlerin kullanım süreleri çok uzun olduğundan dolayı kod üretme mekanizmaları, maliyet-etkin yöntemler olarak düşünülebilir. Örnek vermek gerekirse, 40 yıl önce üretilmiş karo uçakları günümüzde hala kullanılmaktadır. Hatta gerçekleştirilecek modernizasyon faaliyetlerinden sonra 30 yıl daha kullanılmaları hedeflenmektedir. Geçen süreçte kullanılan teknolojiler de değişiyor. Yine örnek vermek gerekirse, daha önceki dönemlerde ADA programlama dili daha popülerken C++ yavaş yavaş ADA'nın yerini almaya başladı. Bundan sonraki dönemlerde C++'ın popüler kalıp kalmayacağını şimdiden tahmin etmek zor olacaktır. Bu nedenden dolayı modelden kod üreten sistemler ilerde yaşanabilecek bir teknoloji dönüşümünde etkili olabilirler.

UGGS Modellemesinde dikkate alınması gereken bir başka konuysa UGGS'nde yaşanabilecek hata durumlarında insan hayatının söz konusu olacağıdır. UGGS güvenlik kritik sistemlerdir. Bu nedenden dolayı UGGS gibi avyonik sistemlerde tüm kod parçacıklarının sertifikasyonunun yapılması gerekmektedir. Sertifikasyonlar değişik kuruluşlar tarafından yapılmaktadır. Örneğin Amerikan Federe Havacılık Dairesi (Federal Aviation Administration – FAA) Amerikan sivil havacılığının, Avustralya Sivil Havacılık Güvenliği Otoritesi (Civil Aviation Safety Authority – CASA) Avustralya sivil havacılığından ve Avrupa Sivil Havacılık Güvenliği Ajansı (European Aviation Safety Agency - EASA) Avrupa sivil havacılığının sertifikasyon işlerinden sorumludur. Bu kuruluşların uluslararası koordinasyonundan birleşmiş milletler ve Uluslararası Sivil Havacılık Organizasyonu (ICAO) sorumludur. Bu kuruluşların temel sorumlulukları arasında sertifikasyon ölçütlerini belirlemek ve bu anlamda gerekli düzenlemeleri yapmak ve politikaları belirlemek vardır. Mevcut avyonik yazılım standartları oldukça katıdırlar. Bu anlamda gösterge sistemleri için tasdik (*certify*) edilebilir kütüphaneler vardır. Örneğin Khronos ES – SC 1.0 tasdik edilebilir bir grafik, OpenGL kütüphanesidir. Birçok UGGS bu kütüphane kullanılarak geliştirilmektedir. Bu çalışmaya başlamada bu kütüphaneye uygun kod üretimi hedeflenmiştir. Ayrıca avyonik alanında kabul görmüş ve RTCA, Incorporated tarafından yayınlanmış olan bir başka standart olan DO178b (Software Considerations in Airborne Systems and Equipment Certification) ölçütlerine mümkün olduğunda uygun kod üretimi hedeflenmiştir. FAA bu standardı avyonik yazılımları sertifiyon standardı olarak görmektedir [1, 2, 6].

Bildirinin organizasyonunu vermek gerekirse, 2.Bölümde UGGS modellemesi vardır. 3.Bölümde UGGS kavramlarının gramere aktarılması yer almaktadır. 4.Bölümde MOF'a dayalı çizimden UGGS üst-modelinin soyut sözdizimi (abstract sytax) ve bu üst-modele örnek somut sözdizimi (concrete syntax) verilmekte ve 5.Bölüm bu üst-modelin kısıtlarını tanımlayan iyi tanımlanmış kurallarla devam etmektedir. 6.Bölüm model dönüşümlerine, modelden model dönüşümüyle girmektedir. Bu bölümde UGGS modelinden GMF modeline dönüşüm anlatılmaktadır. 7.Bölümde kod üretme işlemi yer almaktadır. Son olarak 8.Bölümde UGGS Modellemenin geliştirme süreci tartışılmakta ve 9.Bölümde de sonuç olarak UGGS Modellemenin önemli noktaları verilmektedir.

2. Uçuş Güverte Gösterge Sisteminin Modellenmesi

Uçuş güverte gösterge sistemleri görsel modelleme için uygun sistemlerdir. Bu sistemleri oluşturan bileşenler gerçek hayattaki nesnelerle ifade edilmeye uygundur. Örneğin bir gösterge ve bir hata mesajını görsel olarak ifade etmek zor olmayacaktır. Bu nedenden dolayı, bu alanda görsel modelleme metinsel gereksinimlerden ve tasarımdan daha etkili bir yöntemdir. Ayrıca geliştirilecek MGYG yöntemiyle modelden koda ulaşmak için geçen süreç de kısalacaktır.

Uçuş güverte gösterge sistemleri için soyut sözdizimi oluşturma aşamasında yazarların avyonik alanındaki profesyonel iş tecrübeleri, alanla ilgili yönergeler, hava araçlarıyla ilgili dokümanlar, Avyonik yazılım geliştirmeyle ilgili standartlar incelenmiştir. İncelenen dokümanların başında FAA tarafından yayınlanan 25-11A (ELECTRONIC FLIGHT DECK DISPLAYS – Elektronik Uçuş Güverte Göstergeleri) dokümanı yer almaktadır. Bu doküman uçuş güverte göstergeleri, gösterge elemanları, taşımacılık amaçlı hava araçlarına kurulan ilgili sistemler için tasarım, kurulum, entegrasyon ve onayları için gerekli olan adımlar için yol gösterici niteliktedir [3].

Uçuş güverte gösterge sistemlerinin birçok elemanı vardır. Bu çalışma kapsamında temel elemanlar bir alt küme olarak seçilip modellenmiştir. Seçilen elemanları kullanarak ayrıntılı modellenmeyen elemanlar da geliştirme sürecinde geliştirilebilir. Ayrıca eleman seçimleri geliştiricileri sistem tasarımı, geliştirme etkinlikleri sırasında kısıtlamamaya yönelik de seçilmiştir. Analiz sonrası ortaya çıkan UGGS alan modelinde "ekran (display), semboloji (symbology), metin(text), etiket(label), sembol(symbol), gösterge (symbol)" elemanları yer almaktadır. Terimler sözlüğü Tablo 1'de verilmiştir.

Tablo 1: Uçuş Güvertesi Gösterge Sistemleri için Alan Kavramları

Terim	Açıklama
Ekran (Display)	Ana ekran. Ekran sembolojilerden oluşur.
Semboloji (Symbology)	Ögeleri gruplayan bir yer tutucu.
Metin (Text)	Metinleri tanımlar. Genellikle uyarıları, mesajları ve hataları görüntüler. Üç tip metin tipi vardır: "Uyarı, Normal, Hata".
Etiket (Label)	Etiket başka bir elemanı açıklayan bileşendir. İki tip etiket vardır: "Metin Etiket, Simge Etiket".
Metin Etiket (TextLabel)	Başak bir elemanı açıklayan metin şeklindeki etiket tipidir.
Simge Etiket (IconLabel)	Başak bir elemanı açıklayan simge şeklindeki etiket tipidir.
Sembol (Symbol)	Genel olarak UGGS'nde kullanılan görsel elemanlar için kullanılır. İki tip sembol vardır: "Hava aracı sembolü, yer sembolü".
Hav aracı Sembolü (AircraftSymbol)	Helikopter, uçak gibi hava araçlarını tanımlar.
Yer Sembolü (TerrainSymbol)	Dağ, bina, Vor, havalimanı gibi yerde bulunan şekil ve yapıları tanımlar.
Gösterge (Indicator)	Hız, yakıt, motor sıcaklığı gibi bilgileri göstermek için kullanılır. İki tipi vardır: "Dairesel Gösterge, Çubuk gösterge".
Dairesel Gösterge (Gauge)	Dairesel göstergeler arabalrın analog hız göstegeleri gibidir.
Çubuk Gösterge (Bar)	Çubuk göstergeler bilgileri çubuk şeklide sunan gösterge tipidir.

3. DSL Gramer

Gramer yöntemiyle üst model tanımı geliştirme yöntemleri arasında kıyas yapabilmek adına, UGGS için çıkarılan alan kavramlardan EBNF gösterimi kullanılarak UGGS için DSL (alana özgü dil) gramer tanımlanmıştır. Tanımlanan grameri kısaca açıklamak gerekirse; FDDModel Dispay olabilir. Display birden çok Symbogy içerebilir. Component olarak tanımlanan ifadeyse Text, Label, Symbol, Symbology ya da Indicator olabilir, taban sınıf olarak düşünülebilir. Aynı şekilde Label TextLabel ya da IconLabel, Symbol TerrainSymbol ya da IconLabel, Indicator Gauge ya da Bar olabilir.

Tablo 2: UGGS DSL' için EBNF Gösterimi

FDDModel = Display;

Display = {Symbology};

Symbology = {Component};

Component = Text | Label | Symbol | Symbology | Indicator;

Label = TextLabel | IconLabel;

Symbol = TerrainSymbol | AircraftSymbol;

Indicator = Gauge | Bar;

Terminals are: Gauge, Bar, Text, TextLabel, IconLabel, TerrainSymbol, AircraftSymbol

Non-terminals are: FDDModel, Display, Symbology, Component, Label, Symbol, Indicator

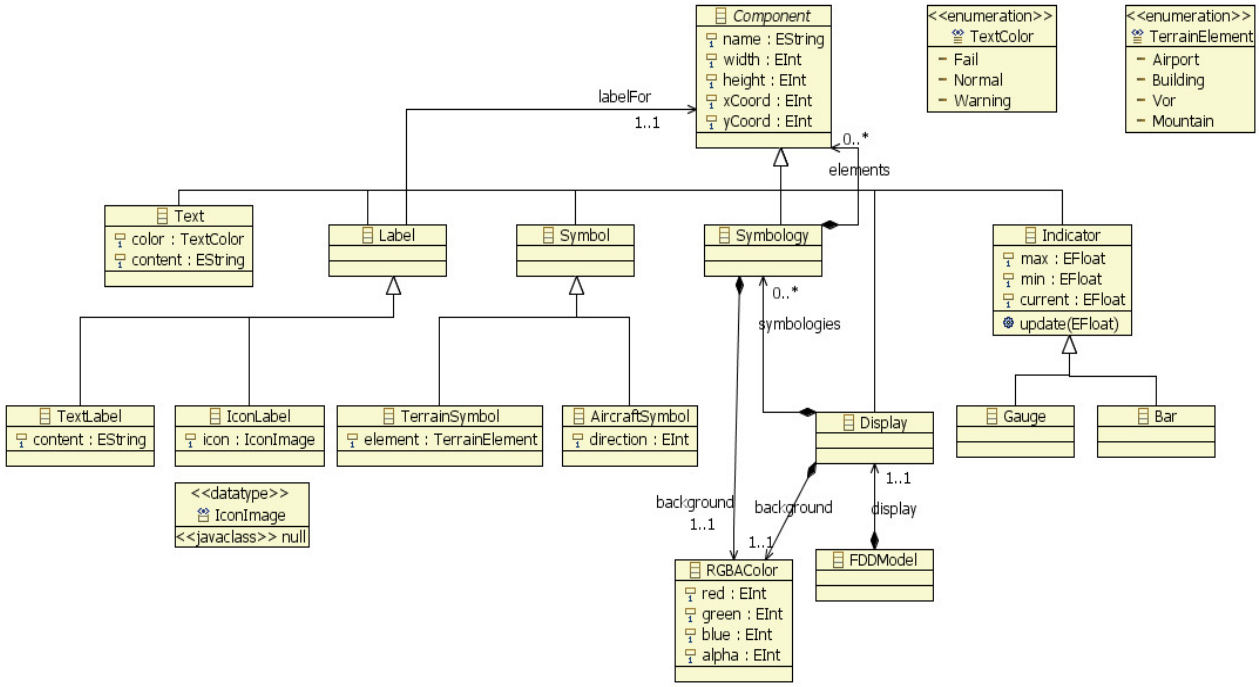
4. MOF Tabanlı Çizimden Üst-model Tanımı

4.1. Çizim Tabanlı Geliştirilmiş Üst-model

UGGS için çıkarılan alan kavramlardan oluşturulan UGGS üst-modeli Şekil 1 verilmiştir. Component bileşeni çıkarılan alan kavramlarına yönelik bir soyutlamadır. Bileşen ismi, bileşenin genişliği, yüksekliği, x ve y düzlemindeki koordinatları gibi temel özellikler Component bileşeninin de verilmiştir. Text bileşenin renk özelliği TextColor tipinde verilmiştir ve TextColor hata, uyarı ve normal durumlar için tanımlar içermektedir. Tablo 1'de de verildiği üzere iki tip Label bileşeni var. İlki içerik tutan TextLabel bileşeni, diğeriyse IconImage tipinde simge tutan IconLabel bileşenidir. Her iki tip Label bileşeni bir Component bileşeninden türeyen bir bileşenle ilişkilendirilmek zorundadır çünkü Label bileşeni ilişkilendirildiği bileşen için açıklayıcıdır. Genel grafikler ve görseller için Symbol

bileşeni tanımlanmıştır ve iki tane tipi vardır. İlki TerrainElement tipinde bir özelliği olan TerrainSymbol bileşenidir. TerrainElement ile havalimanları, dağlar, Vor cihazları tanımlanmaktadır. AircraftSymbol bileşeniyse yön özelliği olmak zorunda olan ve hava araçlarını tanımlayan bir diğer tip Symbol bileşendir. Indicator bileşeni genel sıcaklık, yükseklik, hız, yakıt gibi göstergeleri tanımlayan bileşendir. Bu bileşenin en düşük, en yüksek değerleri tanımlanmalıdır. Yine gösterilen niceliğin mevcut değeri için ve bu mevcut değeri güncellemek için bir metodu olmak zorundadır. Gauge ve Bar bileşenleri sırasıyla dairesel ve çubuk şeklindeki Indicator bileşen tipleridir. Şu ana kadar verilmiş olan özel bileşenler haricinde bu bileşenlerden oluşan Symbology ve Symbology bileşenlerinden oluşan Display bileşenleri de vardır. Bu iki bileşenin ayrıca arka plan renklerleri de RGBAColor tipinde tanımlanmak zorundadır. RGBAColor tipi renk değerini kırmızı, yeşil, mavi ve alfa değerleriyle tutmaktadır. Geliştirilen UGGS modeli üst-modelde FDDModel bileşeni olarak ifade edilmiştir ve FDDModel bileşeninin Display bileşeni tipinde bir elemanı olmak zorundadır.

Üst-model geliştirme ortamı olarak Eclipse IDE ve oAW (openArchitectureWare) araçları kullanılmıştır. Ecore diliyle UGGS üst-modeli oluşturulmuştur. Ecore MOF'un (Meta-Object Facility) basitleştirilmiş bir alt kümesidir. MOF OMG(Object Management Group) tarafından model-güdümlü mühendislikte üst-model geliştirme için geliştirilmiş bir standarttır.



Şekil 1: ECore Üst-üstmodelinden UGGS Üst-modelinin Soyut Sözdizimi Gösterimi

4.2. MOF Tabalı Çizimden Geliştirilmiş Üst-model için Somut Sözdizimi Örneği

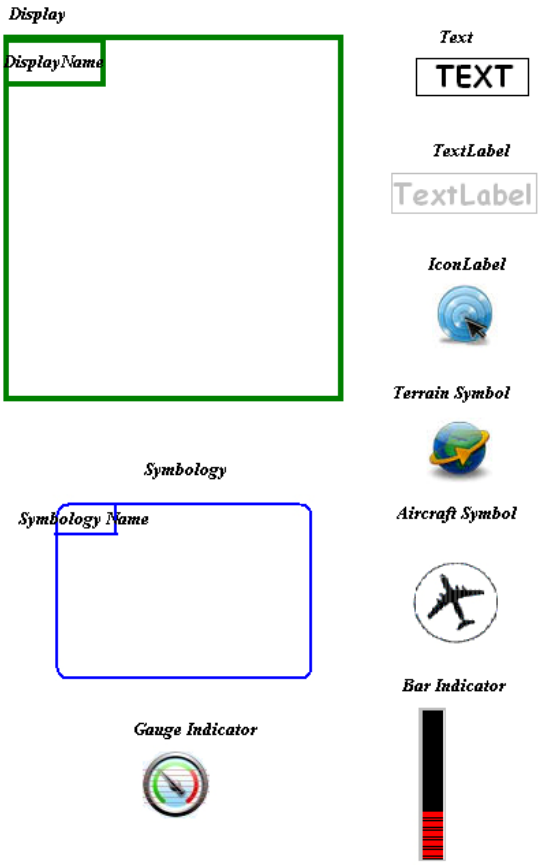
Üst-modelle verilmiş olan soyut UGGS sözdizimi için örnek somut sözdizimi Şekil 2'de verilmiştir. Bu soyut sözdizimi bir grafik programıyla oluşturulmuştur. UGGS üst-modeli için başka örnek somut sözdizimleri tanımlamak mümkündür. UGGS üstmodellemesi verilmiş olan herhangi bir bileşen, somut sözdiziminde görsel herhangi bir şekil tarafından temsil edilebilir. Bu noktada dikkat edilmesi gereken, verilen görsel şeklin ilgili bileşeni çağrıştırmak zorunda olduğudur.

5. Statik Semantikler

UGGS üst-modeli için 15 tane iyi tanımlı kural tanımlanmıştır. Kurallar oAW aracının sağladığı Check Language dilinde tanımlanmıştır. Check Language dilinin sözdizimi JAVA ve OCL (Object Constraint Language) dillerinin sözdizimlerinin birleşimi gibidir [4]. Geliştirdiğimiz UGGS üst-modeli için tanımlanan kurallar Tablo 2'de verilmiştir. Bu kurallar, geliştirilen üst-modelden üretilmiş modellerin, ilgili üst-modele uygunluğunu kontrol ederken kullanılmaktadır. Örneğin ilgili modelden kod üretmeden tanımlanmış, verilen modelin kullara uygun olması gerekmektedir.

Tanımlanmış kurallardan önemli olanlardan birkaçı açıklamak gerekirse; ilk kurala göre FDDModel bileşeninin geçerli bir Display bileşeni olmak zorundadır. İkinci ve üçüncü kurala göre Display bileşenin tüm Symbology bileşenleri ve Symbology bileşenlerinin de tüm Component bileşenleri eşsiz olmalıdır. Dördüncü kuralda, Display bileşenin tüm Symbology bileşenlerinin, yükseklik ve genişlik değerlerinin Display bileşeninkini geçmemesi

gerektiği tanımlanmıştır. Kıyaslama, Symbology bileşenlerinin ilgili değerlerinin toplamlarına göre değil, her birinin değerlerine göre ayrı ayrı yapılmaktadır. Yedinci kuraldaysa hiçbir bileşenin x ve y koordinat pozisyonlarının negatif olamayacağı verilmiştir.



Şekil 2: MOF Tabalı Çizimden Geliştirilmiş Üst-model için Somut Sözdizimi Örneği

Tablo 3: UGGS Üst-modeli için Statik Semantikler

```
context FDDModel ERROR "No Display Defined" :
   display != null;

context Symbology ERROR "All symbologies of Display
have to be unique" :
((Display)this.eContainer).symbologies.select(ele.name ==
this.name) == 1;

context Component ERROR "All elements of Symbology
have to be unique" :
((Symbology)this.eContainer).elements.select(ele.name ==
this.name) == 1;

context Display ERROR "Out of Width" :
this.symbologies.exists(ele.width<=this.width);

context Display ERROR "Out of Height" :
 this.symbologies.exists(ele.height<=this.height);
```

```
context Indicator ERROR "Current Value is Out of Range" :
this.current >= this.min && this.current <= this.max;

context Component ERROR "Invalid X-Y Coordinate":
this.xCoord >= 0 && this.yCoord >= 0;

context Symbology ERROR "Invalid X-Y Coordinate":
this.elements.exists(ele.xCoord<=this.width) &&
this.elements.exists(ele.yCoord<=this.height);

context AircraftSymbol ERROR "Invalid Direction" :
this.direction <= 360 && this.direction >= 0;

context Label ERROR "Label has to be referenced to a
Component" : this.labelFor != null;

context Display WARNING "Background color of Display
has to be more gray" :
this.background.red <= 235 && this.background.green <=
235 && this.background.blue <= 235;

context Symbology ERROR "Sybology cannot have element
at Display or Symbology type" :
this.elements.typeSelect(Display) == false &&
this.elements.typeSelect(Symbology) == false;

context TextLabel ERROR "Text has to be defined for a
TextLabel" : this.content != null;

context IconLabel ERROR "Icon image has to be defined
for a IconLabel" :
this.icon != (IconImage)(null);

context Component ERROR "Name has to defined" :
this.name != null;
```

6. Modelden Modele Dönüşüm: UGGS'den GMF'e

Bu çalışma kapsamında UGGS Modelleme ile görsel modeller yaratmak amaçlanmıştır. Her ne kadar uçuş güverte gösterge sistemlerinin üst-modeli çıkartılmış ve çıkartılan üst-model için örnek somut sözdizimi verilmiş olsa da; ilgili görsel somut sözdizimi bilişenlerinden, model yaratmak için bir araç, zaman kısıtından dolayı geliştirilememiştir. Böyle bir aracın geliştirildiği varsayılarak, böyle aracın ürettiği modeller sadece ilgili araçla uyumlu olacaktır. Bu kapsamda, üretilen aracın çokça kullanılan diğer araçlarla birlikte çalışabilirliği problemi çıkmaktadır. Örneğin, bizim aracımızdan üretilmiş modeller The Eclipse Graphical Modeling Framework (GMF) uyumlu bir araçla açılmak istenebilir. Bu anlamda yapılması gereken modelden modele dönüşüm yapmaktır.

GMF modelleri görselleştirmede geniş kullanım alanı vardır. Örneğin EMF'den (Eclipse Modeling Framework) Ecore uyumlu geliştirilen üst-modelleri görselleştirmek için GMF kullanılmaktadır. Bizim UGGS üst-modelimizde Ecore kullanılarak üretilmiştir. Ayrıca geliştirildiği varsayılan aracımızın UGGS üst-modeline uygun görsel modelleri

ürettiğini varsayarsak, aynı modeli GMF ile de görselleştirebiliriz. Bu açıdan UGGS üst-modelinden üretilen modelleri GMF üst-modeline dönüştürmek anlamlı olacaktır. UGGS'den GMF'e modelden modele dönüşüm ATL (Atlas Transformation Language) kullanılarak sağlanmıştır. ATL modelden modele dönüşüm sağlayacak kurallar tanımlamaya imkân sağlamaktadır [8]. Girdi olarak kaynak üst-model, kaynak model, hedef üst-model verilmekte ve hedef model tanımlanmış kurallara göre üretilmektedir. UGGS üst-modeli zaten tanımlanmıştı. GMF üst-modeli hali hazırda tanımlıdır [7]. ATL sözdizimi kullanılarak UGGS bileşenleri GMF bileşenleriyle eşlenmiştir. Bu kurallar kullanılarak hedef GMF modeli oluşturulabilir.

Sözdizimi renklendirme, ATL dosyasını otomatik derleme gibi kabiliyetlerinden dolayı modelden modele dönüşüm geliştirme ortamı olarak ATL bütünleşik geliştirme ortamı seçilmiştir [8]. Ayrıca ATL'e kolay adapte olmak ve ATL'i kolay kullanabilmek için gerekli dokümantasyonlar mevcuttur. Geliştirilmiş ATL yardımcı kuralları Tablo 4'de verilmiştir. ATL yardımcı kuralları ATL'in metot işlevli yapılarıdır. Şekil 3'de verilmiş olan şekildeki FDD_metamodel UGGS üst-modelini, aFDD ise UGGS modelini teşkil etmektedir. Verilen kaynak modeli dönüştürmek için gerekli olan kurallardan temel olanları Tablo 5'de verilmiştir. Şekil 4'de de dönüşümün çıktısı olan GMF modeli verilmiştir. Dönüşüm şu şeklide olmuştur; Display bileşeni GMF'in Diagram bileşenine, Component bileşeniyse Node bileşenine dönüştürülmüştür. Component bileşeninden dolayı, bu bileşenin alt bileşenleri olan Text, Labels, Indicators, Symbology, Symbol bileşenleri de Node bileşenine dönüştürülmüştür. Bu yüzden Component bileşenin dönüşüm kuralı soyut tanımlanmıştır. Component bileşeninin alt türevlerinin dönüşümleriyle bu soyut kuraldan türemiştir. Ayrıca arka plan renginin tutan özellik "colorvalue" yardımcı kuralı ile örnek bir dönüşüm algoritması kullanılarak dönüştürülmüştür. Biz görsellik sağlayan aracımızı geliştirmediğimizden dolayı GMF'in bağlantıları sağlayan kısımlarına eşlenmeler yapılmamıştır.

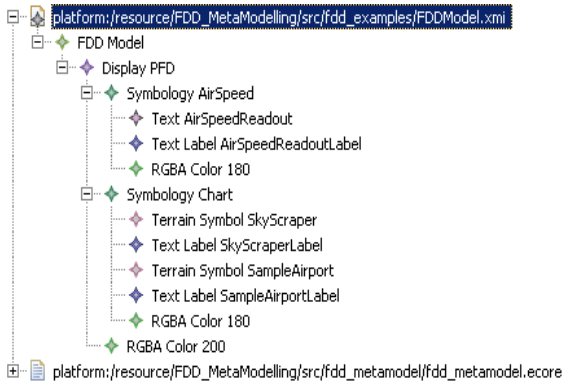
Tablo 4: UGGS'den GMF'e Dönüşüm için ATL Dönüşüm Yardımcıları

```
module FDD2GMF; -- Module Template
create aGMF : GMF from aFDD :
FDD_metamodel;

helper context FDD_metamodel!RGBAColor
def : colorvalue : Integer =
       (self.red*255 + self.green*255 +
self.blue*255 + self.alpha*255);

helper context FDD_metamodel!TextColor
def : normalColor :
FDD_metamodel!RGBAColor =
       Sequence{128, 128, 128, 128};

helper context FDD_metamodel!TextColor
def : failColor : FDD_metamodel!RGBAColor
=
       Sequence{255, 0, 0, 0};

helper context FDD_metamodel!TextColor
def : warningColor :
FDD_metamodel!RGBAColor =
       Sequence{255, 255, 0, 0};

helper context FDD_metamodel!TextColor
def: getRGBAColorFromTextColor() :
FDD_metamodel!RGBAColor =
       if self.color =
FDD_metamodel!TextColor.Normal
       then
          self.normalColor
       else
              if self.color =
FDD_metamodel!TextColor.Fail
              then
                 self.failColor
              else
                 self.warningColor
              endif
    endif;
```

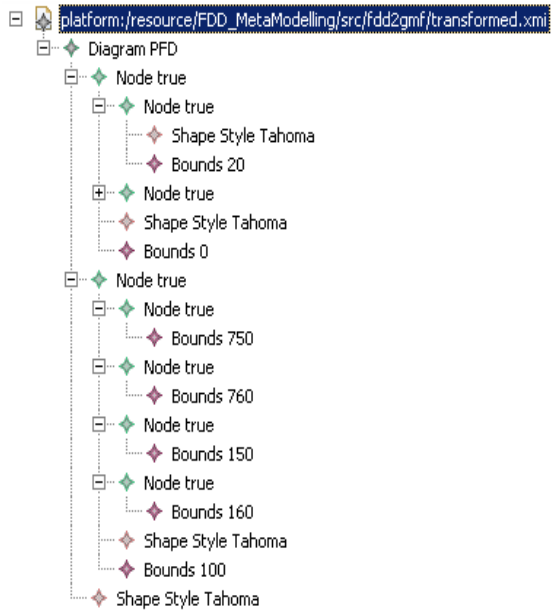
Tablo 5: UGGS'den GMF'e Dönüşüm için ATL Dönüşüm Kuralları

```
rule Display2Diagram
{
from
   display : FDD_metamodel!Display
to
   diagram : GMF!Diagram (
     name <- display.name,
     type <- 'FDD_Display',
     measurementUnit<-#Pixel,
     children <- display.symbologies,
     visible <- true,
     mutable <- false,
     styles <- style,
     layoutConstraint <- size ),
  style : GMF!ShapeStyle (
     fillcolor <- display.background ),
  size : GMF!Size (
     width <- display.width,
     height <- display.height )
}

abstract rule Component2Node
{
from
   component : FDD_metamodel!Component
to
   node : GMF!Node (
     type <- component.name,
     layoutConstraint <- bound,
     visible <- true,
     mutable <- true ),
   bound : GMF!Bounds (
     x <- component.xCoord,
     y <- component.yCoord,
     width <- component.width,
     height <- component.height )
}

rule Symbology2Node extends
Component2Node
{
from
   symbology : FDD_metamodel!Symbology
```

```
to
   node : GMF!Node (
     styles <- style,
     children <- symbology.elements ),
  style : GMF!ShapeStyle (
     fillcolor <-
symbology.background.colorvalue )
}
```



Şekil 3: Örnek UGGS Modeli



Şekil 4: UGGS Model'inden Üretilmiş GMF Modeli

7. Modelden Metne Dönüşüm – Kod Üretimi

Bu çalışma kapsamında, son ürün olarak yeniden kullanılır, tasdik edilebilir, yüksek kalite standartlarına uygun kod üretmek hedeflenmiştir. Avyonik standartların sertifikasyon maliyetleri oldukça fazladır. Hava aracında çalışacak kodun her bir ifadesinin, kodda yer alan karar ve koşulun test edilmesi gerekmektedir. Bu kapsamda ayrıntılı gereksinim bazlı test etkinlikleri ve test sonuçlarının analizlerinin yapılması gerekmektedir. Tüm bu süreç kodda gerçekleşecek herhangi bir değişiklikte, ilgili değişikliklere göre tekrarlanmalıdır. Ayrıca DO178b standardına göre; yazılım gereksinim analizi, tasarım, kodlama ve entegrasyon süreçleri her bir yazılım parçacığı için test etkinliklerine ek olarak uygulanmalıdır. Bu nedenlerle ek olarak modelden kod üretmek iki sebepten dolayı oldukça maliyet-etkin, üretken bir yöntemdir. Bu nedenlerden ilki, kod üretme şablonları bir kez tanımlandıktan sonra, geliştirilen modellerden yüksek oranda otomatik kod üretimi elde edilebilmektedir. Ayrıca tasarım safası güvenlik kritik yazılımlarda yapılması gereken zorunlu bir etkinliktir. Bu nedenden dolayı, modelden tasdik edilebilir kod üretme şablonlarıyla daha kolay ve az maliyetle tasdik edilebilir kod üretimine gidilebilir. Buna ek olarak, modelden koda ulaşılacağı için yazılım geliştirme süreci daha hızlı işleyecektir. İkinci neden olarak, kod üretme mekanizması bakım maliyetlerini düşürecektir. Özellikle yüksek olgunluk seviyesindeki firmalardaki ağır süreçler MGYG ile daha hızlı işleyecektir. Model değiştirilerek koda değişiklikler yansıtılabilecek.

Genel olarak kullanılan iki tip kod üretme tekniği vardır. Bunlardan platform bağımlı ve platform bağımsız modeli metne dönüştürme teknikleridir. Bu çalışma kapsamında platform bağımlı kod üretme tekniği seçilmiştir. Bunda temel etken avynonik görüntüleme sistemlerinde Khronos ES – SC OpenGL kütüphanesinin ve güvenlik kritik gömülü yazılımlarda C/C++ dillerinin yoğun olarak kullanılmasıdır. Ancak ilerleyen yıllarda OpenGL teknolojisi yerini başka bir teknolojiye bırakırsa sorusu akla geliyor. Burada bir seçim yapılmak durumundadır, ya platform bağımsız şablonlardan platform bağımlı şablonlar üretilerek kod üretimine gidilecek ya da platform değişene kadar mevcut platforma özel kod üretim şablonları kullanılacaktır. Hedef alanımızda şu an için kullanılan teknolojiler belli bir olgunlukta olduğundan dolayı platform bağımlı kod üretimi tekniği seçildi.

Çok esnek ve kolay kullanımı olan oAW ile gelen Xpand dili kullanıldı. Xpand ile platform bağımlı (C++ ve Khronos ES – SC OpenGL) modelden metin üretme yoluna gidildi. Uçuş güverte gösterge sistemleri için geliştirilecek uygulama yazılımının ana metodunu üreten şablon tanımı Tablo 6'te verilmiştir.

Geliştirilen kod üretim şablonunda, temel tip tanımları (RGBAColor gibi) haricindeki diğer kod parçacıklarına temel teşkil eden Component üst sınıfı vardır. Ayrıca temel tip tanımları diğer kaynak dosyaları için ortak olan bir kaynak dosyasında üretilir. Sadece modele kullanılmış olan UGGS üst-model bileşenleri için Component sınıfından türetilmiş kaynak kod dosyaları üretilir. Tablo 6'da verilen şablondaki "main" tanımında şöyle bir yapı vardır: "Modelden Display elemanının özellikleri alınır ver bir tane Display nesnesi yaratılır. Daha sonra Display elemanının tüm Symbology elemanları ve her bir Symbology elemanının tüm Component elemanları modelden okunularak yaratılır. Her bir nesne için gerekli kaynak dosyaları da bu süreçte yaratılır. Böylelikte modelden uygulama yazılımı ve kaynak kod üretilmiş olunur.

Component sınıfında "myCode" isimli bir soyut metot bulunmaktadır. Bu sınıftan türetilen sınıflar "myCode" metodunu kendi sorumluluklarıyla ilgili olarak, örneğin gerekli OpenGL API çağrıları ile gerçekleştirirler. Eğer bir sınıfın alt ögeleri varsa, örneğin "Display" sınıfındaki "symbologies" alt ögesi gibi, kendi işiyle ilgili işlemleri yaptıktan sonra alt ögelerinin "myCode" metodunu çağırır.

Bu şekilde sistemin çalışması periyodik olarak "myCode" metotlarının içi içe bir tutumla çağrılmasıyla gerçekleştirilir.

Son olarak, kod üretme şablonunda otomatik üretilen ve elle yazılan kod bloklarını ayırmak gereklidir. Bizim çalışmamızda otomatik üretilen kod blokları daha önceden yazılmış olan kütüphane metotlarını (örnek: OpenGL kütüphane API'leri) çağırmaktadır.

Tablo 6: Xpand Ana Kod Dönüştürme Tanımı

```
«IMPORT fdd_metamodel»
«EXTENSION fdd_template_m2t::GeneratorExtensions»

«DEFINE main FOR fdd_metamodel::FDDModel»
«FILE "FDDModel.cpp"»
#include"Display.h"
#include <iostream>
using namespace std;

int main()
{
bool retVal = true;

Display «display.name» = new Display("«display.name»", «display.width»,
«display.height», (new RGBAColor(«display.background.red»,
«display.background.green», «display.background.blue»,
«display.background.alpha»)));

«FOREACH display.symbologies AS s»
	//Create «s.name» symbology
	Symbology «s.name» = new Symbology("«s.name»", «s.width», «s.height»,
«s.xCoord», «s.yCoord», (new RGBAColor(«s.background.red»,
«s.background.green», «s.background.blue», «s.background.alpha»)));

	«FOREACH display.symbologies.elements AS e»
		//Create «e.name» element
		Component «e.name» = new
«e.metaType.toString().subString(15,e.metaType.toString().length)»("«e.name»",
«e.width», «e.height», «e.xCoord», «e.yCoord»);
		//Add «e.name» symbology to «s.name»
		«s.name».addElement(«e.name»);

		«IF e.metaType.toString().subString
(15,e.metaType.toString().length).matches("Indicator")»
			«EXPAND fddModel2code_classes::Indicator»
		«ENDIF»
		«IF e.metaType.toString().subString
(15,e.metaType.toString().length).matches("Gauge")»
			«EXPAND fddModel2code_classes::Gauge»
			«EXPAND fddModel2code_classes::Indicator»
		«ENDIF»
  «REM»Comment: Other elements can be transformed with similar rules«ENDREM»
	«ENDFOREACH»
	//Add «s.name» symbology to «display.name»
	«display.name».addSymbology(«s.name»);
«ENDFOREACH»

while (retVal == true) {
	retVal = «display.name».myCode();
}
return 0;
}
«ENDFILE»
«EXPAND display_cpp FOR display»
«EXPAND fddModel2code_classes::fdd_common»
«EXPAND fddModel2code_classes::fdd_symbology»
«ENDDEFINE»
```

8. Tartışma

Proje boyunca birçok sorunlar ortaya çıkmasına rağmen MGYG'nin önemi deneyimlenerek öğrenildi. MGYG'nin yararları sınandı. Bu yararlar alt konulara göre aşağıda verilmiştir.

8.1.Araçlar

Kullanılan araçlardan kaynaklan birçok engel ile proje boyunca karşılaşılarak tecrübe edilmiştir. Özellikle oAW birçok hata içermekte ve buda üretkenliği düşürmektedir. Model – görsel model uyumluluğu değişiklik yapıldığı durumlarda kaybolabilmektedir. Olgunlaşmamış bu araçlar bazı sorunların çözümü için sizi zorlamakta ve zaman kaybına sebep olmaktadır. Ayrıca, oAW'nin standartları desteklememesi kullanıcıda büyük bir endişeye sebep olmaktadır. Diğer bir ifadeyle ne Ecore ne de Check Language bir standarttır. Proje başında MOF ve OCL kullanımı amaçlanmıştı ancak oAW çatısı bunları desteklememektedir. Diğer bir önemli husus ise bu araçların iyi belgelendirilmiş olmamasıdır. Buna rağmen Ecore geliştirmek için kullanılan görsel kullanıcı arayüzü öğrenme sonrası çok verimli ve faydalı olmaktadır.

ATL dönüşüm kuralları tanımlamak için gayet esnek bir dildir. Ancak, Eclipse ortamında ATL ile çalışırken birçok hata ortaya çıkabilmektedir. Örneğin, ATL dosyası derleme konusunda bir çok sorunla karşılaşılmıştır. Bu durumla karşılaşıldığında çözüm için ATL projesini tekrar ve tekrar oluşturmak gerekti. OAW Xpand de modelden metin dönüşümü sağlayan çok esnek bir dildir.

8.2.Gramer

Gramer kullanımı Alan Özelleşmiş Dil (DSL) tanımlamak için verimli ve uygun bir yöntem değildir. EBNF simgelimi daha çok çözüm (diğer bir deyişle programlama dili) alanı için uygundur. Kısıtlar ve ilişkiler açıkça ifade edilememektedir. Ayrıca belirsizliklere açık bir yapısı vardır. EBNF yerine MGYG için daha anlamlı üst-söz dizimi tanımlanabilir.

8.3.Sıfırdan Üst-model Tanımlama

Sıfırdan üst-modelleme gramer tanımlamaya göre daha kolaydır. Aslında, projede üst-model önce tanımlanmış ve buradan esinlenerek gramer oluşturulmuştur. Üst-modelleme çok daha anlamlı ve tanımlayıcıdır. İç tipler ve soyutlamalar kullanılabilmektedir. Üst-modelleme ile kısıtlar ve ilişkiler açıkça tanımlanmaktadır. Ayrıca üst-modelleme süreci M1 (model) ve M2 (üst-model) ögelerini birbirinden ayırabilmenize olanak sağlar. Bize göre üst-modelleme MGYG yaklaşımı için en iyi yoldur.

8.4.Alanın Soyut Sözdizimi için Somut Sözdizimi Geliştirme

Grafiksel düzenleyici geliştirmek için, GMF (The Eclipse Graphical Framework) tarafından EMF (Eclipse Modeling Framework) ve GEF (Graphical Editing Framework) tabanlı üretken bir çalışma altyapısı sağlanmaktadır [5]. Belli bir alan için GMF kullanılarak grafiksel bir düzenleyici geliştirmenin çok fazla zaman kaybına sebep olacağı anlaşılmıştır. Bu yüzden UGGS alanı somut sözdizimi için grafik programları kullanılmıştır.

8.5.MGYG

MGYG üretkenliği artıran verimli bir yaklaşımdır. Problem alanına yoğunlaşarak özel bir alan için çözüm üretimi sağlamaktadır. Alan özelleşmiş üst-model bir kez oluşturulduktan sonra problem alanının çözümüne yönelik modeller kolaylıkla üretilebilmektedir. MGYG'nin ana yaklaşımı olan çalıştırılabilir model sayesinde üretkenlik yükselmektedir. Ayrıca, üst-model üretilen tekrar kullanılabilir ögeler sayesinde MGYG tekrar kullanılabilirliği artırmaktadır.

8.6.Modelden Model Üretimi

Modelden model dönüşümü özellikle sıfırdan üst-modelleme yapıldığı zaman gerekli olmaktadır. Çünkü modelden modele dönüşümler tanımlanmaz ise geliştirilen modeller yoğunlukla kullanılan UML, GMF gibi araçlarla birlikte çalışamazlar.

8.7.Modelden Kod (Metin) Üretimi

Metin üretme aşaması standart yazılım mühendisliği aşamalarından olan tasarım ve kodlama safhalarının birleşimi şeklinde tanımlanabilir. Xpand çok etkili bir şablon temelli metin üreticisidir. Dosya üzerinde katar işlemleri, model elemanlarına dayanan değişken isimleri gibi çok çeşitli üretim kuralları tanımlamaya izin vermektedir. Ayrıca kontroller (statik semantikler) Xpand tarafından çalıştırılabilmekte ve girdi modele göre dosya üretimini mümkün kılabilmektedir.

9. Sonuçlar ve Gelecek Çalışmalar

Bu çalışma kapsamında, UGGS için oluşturulan üst-modele dayanan modeller üreten bir araç geliştirmesi hedeflendi. Çalışma sonunda UGGS üst-modelinden oAW aracı ile model üretebilir, üretilen modelden otomatik olarak Khronos ES – SC OpenGL uyumlu C/C++ kodu üretebilir ve UGGS üst-modelinden üretilen modelleri ATL ile GMF modeline çevirebilir bir mekanizma geliştirilmesi başarıldı. Ama kısıtlı zaman nedeniyle görsel UGGS modelleri oluşturan bir araç geliştirilmedi. Geliştirilen mekanizmaya uygun bir araç geliştirildikten sonra tasdik edilirse, daha kolay ve daha az maliyetle tasdik edilebilir yazılımlar üretmek mümkün hale gelecektir. Böylece yazılım geliştirme maliyetlerinde önemli bir düşüş sağlanmış olacaktır.

Bu çalışma kapsamında geliştirilen platform bağımlı kod üreteme şablonları ile UGGS modellerinden nerdeyse 100% oranında kod üretilmesi başarıldı. Üretilmeyen kodlar kısımları ise kullanılan kütüphaneler oldu. OpenGL kütüphaneleri manüel kod parçacıklarımız oldu. Gelecekte OpenGL yerine The Microsoft DirectX® teknoloji kullanılacak olduğu takdirde, modellerde herhangi bir değişiklik gerekmeden, sadece kod üretme şablonlarındaki OpenGL bağımlı kısımları değiştirmek yürütülür koda ulaşmak mümkün olacaktır.

Görsel modelleme yapabileceğimiz bir araç geliştirmediğimiz halde Eclipse EMF çatısı ile gelen Ecore Model Editor ile örnek modeller geliştirildi. UGGS üst-modelinden ilgili editör sayesinde çok hızlı bir şekilde modeller üretilmiştir. Üretilen modellerden tanımlanmış metin üretme şablonları sayesinde birkaç saniye içerisinde tüm uygulama kodu üretildi. Geliştirilen yazılım geliştirme ortamı sayesinde sürekli canlı kalan ve uzun kullanım süreleri olan avyonik ürünlerinin yazılım kısımları, gelecekteki teknolojilere çok hızlı uyum sağlayabilir hale gelecektir.

Gelecek çalışma olarak görsel modelleme için bir araç geliştirilebilinir. Böylesi bir araç sayesinde soyutlama seviyesinde de artış olacaktır. Böylece yazılım geliştirmede karşılaşılan problemin karmaşıklık seviyesinde düşüş sağlanacaktır.

Kod üretme şablonları çok daha kaliteli ve güvenlik standartlarına çok daha uygun kodlar üretilecek şeklide iyileştirilebilir. Bu çalışma kapsamında UGGS modellerinden çok etkin ve esnek kod üretebileceğini gösterildi. Gelecek çalışma olarak, modelden üretilen kod birimleri için test adımları üretmek üretkenlikte artış sağlayacaktır.

Çalışmayı ve sonuçlarını bir bütün olarak düşünmek gerekirse, uçuş güverte gösterge sistemleri model güdümlü yazılım geliştirme için uygun bir alandır. Seçilmiş bir alan için model güdümlü yazılım geliştirme ortamı oluşturmayı (seçilen alanın analizi, alan için üst-model bileşenlerini ve aralarındaki ilişkileri tanımlamak, model dönüşüm kurallarını tanımlamak) yatırım olarak görmekte fayda vardır. Standard yazılım yaşam döngü süreçlerinin uygulama maliyetlerinde düşüş sağlamaktadır; çünkü modelden koda ulaşılmakta ve değişen gereksinimleri yazılıma yansıtma sürecinde etkili bir yöntemdir. MGYG'yi UGGS ve türevleri güvenlik kritik sistemlere bir bütün olarak ve etkin bir şekilde uygulandığı takdirde; üretkenlikte ciddi artışlar, geliştirme ve bakım maliyetlerinde düşüş sağlayacaktır.

10. Teşekkür

Yrd. Doç. Dr. Bedir Tekinerdogan'a bu çalışmadaki çok değerli önerileri ve katkıları için teşekkür ederiz.

11. Kaynakça

[1] RTCA, Inc., *Do178B Software Considerations in Airborne Systems and Equipment Certification*, USA

[2] U.S. Department of Transportation Federal Aviation Administration, Advisory *Circular 20- 115B – RTCA, Inc., Document RTCA/DO-I 78B*, USA, l/ll/93

[3] U.S. Department of Transportation Federal Aviation Administration, *Advisory Circular 25-11A – Electronic Flight Deck Displays*, USA, 6/21/07

[4] *openArchitectureWare User Guide*, Version 4.3.1, http://www.openarchitectureware.org'den ulaşılabilinir, , 17/04/2009 tarihinde erişildi

[5] *Eclipse Graphical Modeling Framework*, http://www.eclipse.org/modeling/gmf'de mevcuttur, 17/04/2009 tarihinde erişildi

[6] http://en.wikipedia.org/wiki/DO178B, 17/04/2009 tarihinde erişildi

[7] http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.gmf.doc/prog-guide/runtime/Developer%20Guide%20to%20Diagram%20Runtime.html, 16/05/2009 tarihinde erişildi

[8] http://www.eclipse.org/m2m/atl/, Accessed at 16/05/2009

[9] *Architecture Board ORMSC, Model Driven Architecture (MDA)*, Document number ormsc/2001-07-01, 9/7/ 2001

[10] http://www.omg.org/technology/documents/modeling_spec_catalog.htm, 15/04/2009 tarihinde erişildi