KABAN-2: Kullanıcı Ara Birimi ve Geçerleme Altyapısı

Atakan Şimşek, İbrahim Demir, Zahir Tezcan

TUBİTAK BİLGEM İLTAREN, Ankara
e-posta:
atakan.simsek@tubitak.gov.tr
ibrahimdemir.mail@gmail.com
zahir.tezcan@tubitak.gov.tr

Özetçe. Simülasyon projeleri gibi verilerin önem arz ettiği, sistem ve arayüz parametrelerinin karmaşık olduğu ve parametrelerin bir birini etkilediği uygulamalarda, verilerin sistematik ve etkin bir geçerleme sürecine tabi tutulması vazgeçilmez bir önkoşuldur. Veri parametrelerinde yapılan bir yanlışlık sistemin yanlış çalışmasına sebep vermekte ve uygulamanın karmaşıklığı ile doğrusal orantıda hatanın kaynağını bulma ve hata çözme maliyetine sebep olmaktadır. Bu yayında, projelerdeki girdi uzayını, hataya sebebiyet vermemek için dinamik olarak geçerleyen, geçerlediği veri için otomatik arayüz üreten, ürettiği arayüzde programın doğru şekilde kullanımını zorunlu kılan(geçerleme sistemi sayesinde) KABAN altyapı projeleri anlatılmaktadır. Kurumumuzda yıllarca kullanılan KABAN-1 altyapısı, projelerdeki tecrübeler ve ihtiyaçlar doğrultusunda geliştirilip KABAN-2 projesi halini almıştır. Bu yayın ikisini de kapsamaktadır.

Anahtar Kelimeler: Dinamik geçerleme, veri güvenilirliği, otomatik arayüz oluşturma, semantik doğrulama

1 GİRİŞ

Veri kalitesinin ve doğruluğunun önem arz ettiği projelerde kullanıcıdan tutarlı veri girdisi sağlamak hayati öneme sahiptir. Karmaşık girdi uzayı olan projelerde kullanıcı ne kadar yetkin bile olsa dikkatsizlik sonucu ya da hesap hatalarından dolayı yanlış veri girdisi programın hatalı sonuç üretimine ya da çalışmamasına sebep verebilmektedir. Çalışma süresi saatler ya da günlerle ifade edilen projelerde her bir hatalı çalıştırma ciddi zaman ve maliyet problemleri çıkarabilmektedir. Daha da kötüsü eğer hatalı girdi kümesi fark edilmezse ve bu analizin sonuçları insan hayatını ilgilendiren güvenlik-kritik projelerde kullanılacaksa insan hayatını tehlikeye atacak sonuçlar ortaya çıkabilmektedir.

Uzun soluklu projelerde, genellikle ister dokümanı ne kadar iyi tanımlanmış olursa olsun ihtiyaç makamının istekleri zamanla şekillenebilmektedir. Değişen isterler kapsamında arayüzün yeniden tasarlanması ihtiyacı doğabilmektedir. Yeni girdilerin eklenmesi, parametre gruplarının değişmesi, en azından parametrelerin yerinin değişmesi projelerde sık karşılaşılan durumlar arasındadır. Bu durumlarda çoğu kez

geçmişte yapılan işlerinin bir kısmının kullanılamaması ve zaman-maliyet yükü gibi sonuçlar ortaya çıkmaktadır.

Yukarıda bahsedilen problemlere çözüm olması adına TÜBİTAK-İLTAREN olarak bir altyapı projesi geliştirilmiştir. KABAN-1 ismi verilen bu projede dinamik veri geçerleme bileşeni ve otomatik arayüz oluşturma bileşeni bulunmaktadır. Bu altyapı yıllarca değişik benzetim projelerinde kullanılmıştır. Zamanla yeni istekler oluşmuş ve gelişen teknoloji sayesinde yeni kabiliyetler eklenme imkânları ortaya çıkmıştır. Bu sebeple KABAN-2 projesi geliştirilmiştir. Bu yeni proje sayesinde hem daha önce kullanılan projenin tecrübelerinden faydalanılmış hem de daha kullanışlı, daha kabiliyetli bir altyapı projesi oluşturulmuştur.

Bu yayının geri kalanı aşağıdaki gibi düzenlenmiştir. İkinci bölümde bu problemlere çözüm olarak başka sistemler ve nasıl çözümler kullanılmış onlar incelenecektir. Üçüncü bölümde bizim çözümümüz olan KABAN-1 sisteminin ana bileşenleri ve yapısı anlatılacak. Aynı bölümde buna ek olarak KABAN-1 sisteminin eksiklikleri irdelenip çözüm olarak geliştirilen KABAN-2 sisteminde bu eksiklikler nasıl giderildiği konusu incelenecektir. Dördüncü bölümde sonuçlar üzerinde konuşulacaktır.

2 İLGİLİ ÇALIŞMALAR

Verinin bütünlük ve doğruluk arz etmesi neredeyse bütün projelerde istenilen bir önkoşuldur. Eğer güvenlik-kritik projelerden bahsedersek bu önkoşul çok daha hayati bir önem göstermektedir. Çünkü yanlış girdinin kullanıldığı bir senaryoda yanlış çıktı üretilebilir ve daha kötüsü insan hayatına mal olabilecek hatalar yapılabilir. Buna engel olmak adına girdi uzayının doğruluğunu geçerleyen sistemler geliştirilmiştir. "Commons Validator"[2], "DataSift"[3] ve "iScreen"[4] bu bağlamda örnek verilebilecek sistemlerdir. Bu sistemler ayrıntılı olarak incelenmiş, artıları eksileri detaylı şekilde analiz edilerek kendi geliştirdiğimiz altyapılarda bu tecrübelerden faydalanılmıştır. TÜBİTAK-İLTAREN tarafından geliştirilen KABAN sistemlerini bu sistemlerden ayıran temel özellikler ise kural ifade gücü, dinamik geçerleme kabiliyeti ve otomatik arayüz üretme kabiliyetidir [1].

Kural ifade gücü geçerleme sistemlerinin omurgasını oluşturmaktadır. Bu sebeple KABAN sistemlerinin kural tanımlama altyapısı için titizlikle davranılıp, akademik çevrelerde kullanılan kural tanımlama dilleri incelenmiş ve bunlar arasında en ileri olduğu gözlemlenen NxBre[8], CLIX[13], RuleML[14,15], ARML[12], Starburst[11], Ariel[9,10] detaylı şekilde analiz edilmiştir. İlgili çalışmaların artıları ve eksileri irdelenerek KABAN kural çalışma dili bu veriler ışığında tasarlanmıştır.

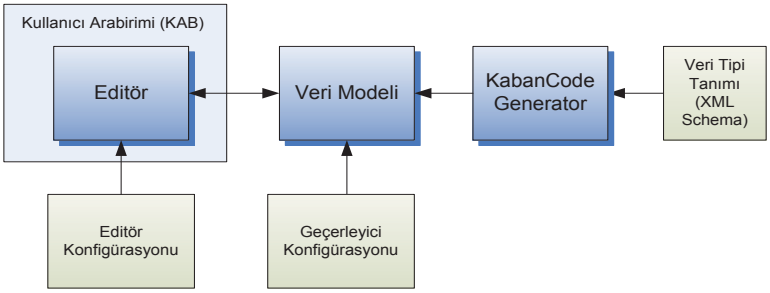
3 ÖNERİLEN ÇÖZÜM

Yukarıda bahsedilen problemlere çözüm olması için geliştirilen KABAN altyapı projeleri, girdi uzayını dinamik bir şekilde geçerlemeye tabi tutmaktadır. Geliştirilen altyapı; veri yapıları için aralık değer kontrolü, format kontrolü gibi basit geçerlemeleri yapabilmenin yanı sıra; birden fazla veri yapısı arasındaki semantik bağlantıyı kontrol ederek veri yapılarını bağlı olduğu girdilerin değerlerine göre dinamik doğru-

lama işlemi de yapabilmektedir. KABAN sistemlerinin kabiliyetleri sadece bunlarla da sınırlı değildir. Eğer bir girdinin değerinin başka girdilere bağlı şekilde hesaplanması gerekiyorsa, sistem bunu dinamik ve otomatik bir şekilde hesaplayıp arayüzde ilgili yere koyabilecek kabiliyete sahiptir. Eğer bağlı olunan parametrenin değeri değişirse, altyapılarımız bu hesaplamaları yeniden yapıp her zaman doğru değeri gösterecek şekilde tasarlanmıştır.

Geliştirdiğimiz altyapıların bir diğer özelliği de yukarıda bahsedilen geçerleme sisteminin arayüzünü otomatik olarak oluşturmasıdır. Böylelikle arayüzü ya da girdi uzayı sık sık değişen projelerde, yeniden arayüz tasarım maliyetinden kurtarmaktadır. Arayüzün otomatik oluşturulması sistemin sağladığı bir ek özellik olması itibariyle istem dâhilinde ilgili yapılandırma dosyası sayesinde arayüz tasarımına müdahale edilmesine imkân sağlanmaktadır. İstek dâhilinde arayüz üretimi devre dışı bırakılıp, geliştirici tarafından geliştirilen arayüzü sisteme kolay bir şekilde bütünleştirilmesine imkân vermektedir. Bu bölümün geri kalanında çözüm olarak ürettiğimiz iki altyapı projesi anlatılacaktır.

3.1 KABAN-1



Şekil 1. KABAN-1 Mimari Tasarım

Şekil 1 de görüldüğü üzere KABAN-1 altyapısı[1] temel olarak üç ana parçadan oluşmaktadır. Bu bileşenleri alt başlıklarda detaylı olarak inceleyecek olursak:

3.1.1 KabanCode Generator Bileşeni:

Mantık olarak Microsoft Visual Studio içinde bulunan "XSD.exe" ile aynı olmasına rağmen "XSD.exe" de bulunan bir eksikliğin tamamlanması maksadıyla geliştirilmiştir. "XSD.exe" de eksiklik sadece isteğe bağlı alanlar için "Specified" alanını oluşturmasıdır. KABAN-1 sisteminin yapısı gereği herhangi bir alanın içindeki değerin tutarlı bir değer olup olmadığının bilinmesi gerekmektedir. Bu yüzden geliştirilen KabanCodeGenerator, tüm alanlar için "Specified" isminde tipi "bool" olan bir alan ekler. KABAN sisteminde veri yapıları XML şema tipinde tanımlanmaktadır. KabanCodeGerator bileşeni "Xml Schema" nesnelerini okuyarak, C# sınıf kütüphanelerini oluşturur.

3.1.2 Veri Modeli Bileşeni:

Bu bileşen sınıf kütüphanelerini okuyarak veri modellerini oluşturur. Veri modelleri hiyerarşik bir yapıda oluşturulduğundan, bu yapı veri model ağacında serbest gezinme imkânı vermektedir. Bu tasarım KABAN-1'in geçerleme kabiliyetinin gücünü artırmaktadır. Karmaşık geçerlemelerde veri model ağacı ilgili alanlar için gezilerek, dinamik hesaplama yapılabilmektedir.

Geçerleme kuralları Veri Modeli bileşenine bir yapılandırma dosyası sayesinde yüklenmektedir. Bu dosyayı geliştirici bir seferlik oluşturmakta ve geçerleme kuralları değiştiğinde sadece ilgili alanının geçerleme kuralını değiştirerek sistemi yönetmektedir. Bu dosya XML tabanlı ve doğruluğu XML şema tarafından doğrulanacak şekilde tasarlanmıştır, bu sebeple kopyala-yapıştır, araya elaman ekleme, aradan eleman silme işlemleri kolaylıkla yapılabilmektedir.

KABAN altyapı projelerinin geçerleme sisteminin gücü, kural tanımlama şemasının ifade gücüne dayanmaktadır. KABAN geçerleme sistemi basit aralık geçerlemelerinden, karmaşık bağıntısal geçerlemelere kadar kullanıcıya lazım olabilecek çok geniş bir aralık sunmaktadır. Her ne kadar kural tanımlama altyapısının geliştiriciye lazım olabilecek bütün geçerleme tiplerini kapsadığı düşünülse de, her ihtimale karşı KABAN sistemi C# dilinde yazılmış kod bloğunu geçerleme sistemiyle bütünleşmesine imkân sağlayacak altyapıyı da içinde barındırmaktadır. Uygulama kullanıcısı istediği an geçerleme için C# bloğunu kullanabilmektedir.

```
<Expression>
      <ArithmeticOp ArithmeticOperationType="subtract">
         <item>
            <containerReference containerReference="..\SinifKapasitesi"/>
         </item>
         <item>
            <ListCount containerReference="..\OgrenciListesi"/>
         </item>
      </ArithmeticOp>
</Expression>
```

Yukarıda, KABAN-1 sisteminden alınan parametre değerleri birbirine bağlı bir geçerleme örneği verilmektedir. Bu örnekte bir üniversite sınıfındaki boş öğrenci kontenjanının sınıfın kapasitesinden öğrenci listesindeki öğrenci sayısının çıkarılması ile hesaplanmasını tanımlayan geçerleme parçacığı görünmektedir.

Eğer bir veri modeline bağlı geçerleyicide hata hesaplanır ise ağaç yapısı sayesinde bu hata en tepe elemana kadar çıkarılabilir. Bunun sayesinde en dip elemanda (belki de o sırada arayüzde görünmeyecek kadar aşağıda bulunan bir elemanda) bile hata olsa bu hata yukarı kadar taşınacağından kullanıcı uyarılmış ve kullanıcının gözünden kaçmamış olur.

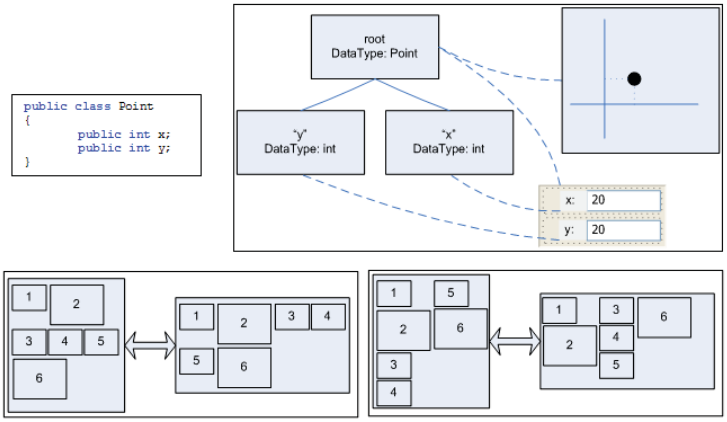
3.1.3 Editör Modülü

Editör bileşeni ilgili yapılandırma dosyasını okuyarak, otomatik arayüz oluşturma işlevini görmektedir. Bu sayede arayüzü sık sık değişen projelerde arayüzün tekrar

tekrar yapılması maliyetinin önüne geçilmektedir. Arayüz yapılandırma dosyası da XML tabanlı bir dosyadır, bu sebeple dosya güncelleme işlemleri kolaylıkla yapılabilmektedir.

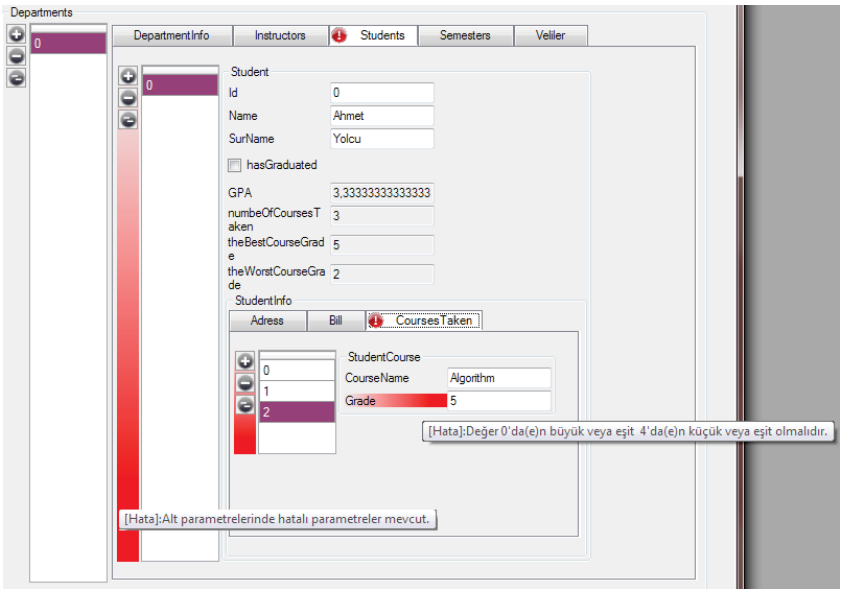
```
<Editor dataType="DemoStruct" name="DemoEditor" isDefault="true" labelVisibility="NoLabel">
  <BasicCustomEditor>
    <Children>
      <TypedChild dataType="D1GuiStruct" fieldRef="D1" column="0" row="0"/>
      <TypedChild dataType="D2GuiStruct" fieldRef="D2" column="1" row="0"/>
      <TypedChild dataType="D3GuiStruct" fieldRef="D3" column="2" row="0"/>
    </Children>
  </BasicCustomEditor>
</Editor>
```

Yukarıdaki kod bloğunda, grup halinde gösterilmesi gereken bir yapı ve onun alt parametrelerini gösteren bir yapılandırma parçası görünmektedir. Burada da görüldüğü gibi kullanıcının sadece genel dağılımı vermesi KABAN sistemleri için yeterlidir. Ama kullanıcı kendi istekleri doğrultusunda daha detaylı tanımlamak isterse yapılandırma dosyasında tanımlanan diğer parametreleri de kullanabilir. Eğer kullanıcı bunların da yeterli olmadığını düşünürse C# kullanarak editörü tanımlayabilir. KABAN, C# sınıflarını editör bileşeninde direk kullanabilecek şekilde tasarlanmıştır.



Şekil 2. Editör dizilim örneği

Şekil 2 de KABAN-1 editör bileşeninin girdileri arayüzde nasıl dizdiğinin bir örneği gösterilmektedir. Resmi dört parçaya bölersek, sol üst kısımda Point isminde bir sınıf tanımlanmış ve bu sınıfın iki alt girdisi bulunmaktadır. Resmin sağ üst kısmında Point sınıf elemanlarının gerçek arayüzde karşılığı gösterilmektedir. Resmin sol alt kısmında kullanıcı kontrolünde yatay akışkan dizilimi, resmin sağ alt kısmında ise resmin dikey akışkan dizilimi gösterilmektedir. Geliştirici bu dizilimler içinde istediği yerleşimi yapılandırma dosyasında sadece birkaç parametre değerini değiştirerek elde edebilir.



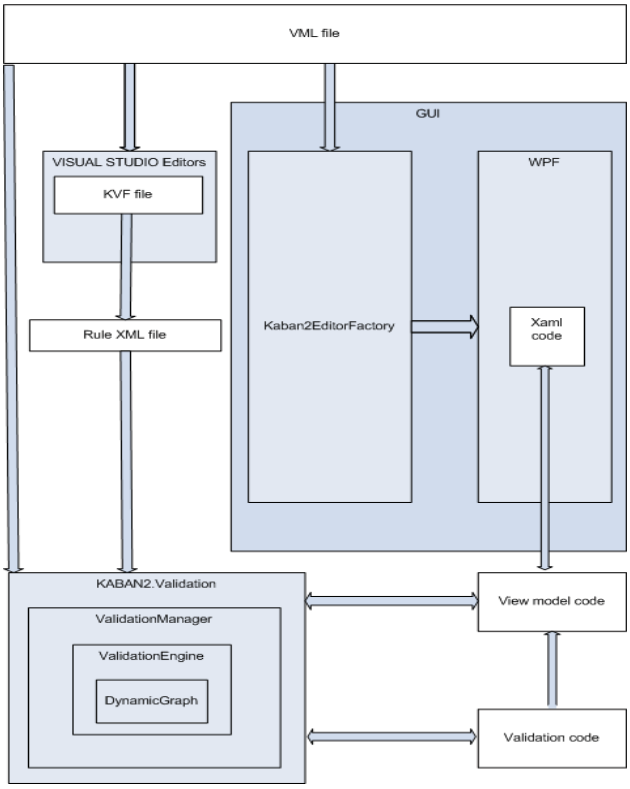
Şekil 3. KABAN-1 sistemi örnek ekran görüntüsü

Şekil 3 de KABAN-1 sistemi kullanılarak oluşturulan bir demo projesinin ekran görüntüsü verilmiştir. Değinilen projede çok basit haliyle bir üniversitenin öğrenci işleri programı yapılmaya çalışılmıştır. Arayüz tamamen otomatik üretilmiş, herhangi bir ek tanımlama yapılmamıştır. Burada "Courses Taken" sekme grubu altında "Grade" girdisinde bir kırmızılık bulunmaktadır. Bu kırmızılık parametrenin geçerlemesinde hata olduğunu göstermektedir. Sekme alanının başlığında ve listenin üzerinde kırmızılık olması da alt parametrelerin birinde hata olduğunu belirtmektedir. Eğer bir parametrede hata varsa, o parametreden en tepe parametreye kadar geçerleme ağacındaki her alan için hata ışığı yakılır. Tabi ki bu özellik seçmelidir ve kapatılabilir. Bu özelliğin amacı herhangi bir alandaki geçerleme hatasının gözden kaçmaması ve sistemin bütün parametrelerin geçerlemeyi geçtiğini garanti etmektir. Hatayı mesajını görmek için kullanıcının fare ile kırmızı yerin üzerine gelmesi gerekmektedir.

3.2 KABAN-2

Zaman içinde kullanılan projelerde yeni istekler oluşmuş ve KABAN-1 altyapısında bu ihtiyaçların giderilmesi zorlaşmaya başlamıştır. Bu sebeple KABAN-1 projesindeki tecrübeler kullanılarak yeni bir altyapı projesi geliştirilmiş, önceki ile aynı çalışma mantığına sahip olduğu için KABAN-2 şeklinde isimlendirilmiştir.

3.2.1 Mimari Yapı



Şekil 4. KABAN-2 Mimari Yapı

KABAN-2 projesi KABAN-1 projesinin gelişmiş halidir bu sebeple KABAN-1 projesinde anlatılan her kabiliyeti destekleyecek şekilde tasarlanmıştır. Çalışma mantığı ve kabiliyet açısından aynı özelliklere sahip olduğundan yukarıda anlatılan kabiliyetler bu bölümde tekrar anlatılmayacaktır.

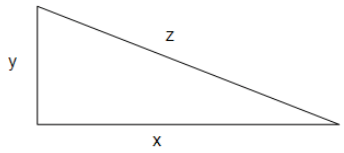
KABAN-2 projesinin mimari yapısı KABAN-1 den farklı bir görünüme sahiptir çünkü MVVM deseni üzerine inşa edilmiştir. Geliştirilen "view model" kodları MVVM uyumludur. Geliştirici VML dosyası kullanarak view modelleri tanımlayabilmektedir. Geçerleme kuralları KVF dosyaları vasıtasıyla tanımlanabilmekte, arayüz de otomatik "Xaml" dosyaları sayesinde oluşturulmaktadır. VML ve KVF dosyaları bizim tarafımızdan geliştirilen dosya formatlarıdır, "Xaml" dosyaları da KABAN-2 projesine bütünleşik çalışan bir uygulama sayesinde otomatik oluşturulmaktadır.

KABAN-2 projesi çalışma amacı açısından KABAN-1 ile aynı kıstaslara sahiptir, amacı dinamik geçerleme yapmak ve otomatik arayüz üretmektir. Mimari yapısı ve kullandığı teknolojiler farklıdır, bu sebeple KABAN-1 e göre daha hızlı çalışmaktadır ve geliştirici açısından kullanımı daha kolaylaştırılmıştır.

3.2.2 Geçerleme Optimizasyonu

KABAN-1 sistemi arayüzdeki herhangi bir parametrenin değeri değiştiğinde o alana bağlı bulunan parametrelerinin değerini gerçek zamanlı yeniden hesaplayacak şekilde tasarlanmıştır. Sistem büyük benzetim projelerinde(parametre sayısı > 500) test edilmiştir, gerçek zamanlı değer değişikliğinin kullanıcının fark edemeyeceği kadar küçük zaman değişiminde yapıldığı görülmüştür fakat dosyadan senaryo yükleme, listelere büyük veri ekleme ya da silme gibi işlemlerde kullanıcının fark edebildiği bir gecikme yaşandığı gözlemlenmiş ve sistemin en uyumluluğu açısından incelenmiştir. Sonuç olarak KABAN-2 altyapısında sıralanmış çizge yapısının sistem ilk ayağa kalktığında sistem tarafından oluşturulmasına ve değişen değerlere göre bağımlılıkların bu çizge yapısına başvurularak yeniden hesaplanmasına karar verilmiştir.

Örnek verilecek olursa:



Şekil 5. Dik üçgen tanımı

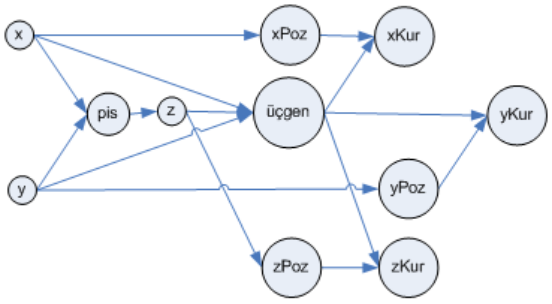
Şekil 5'de verilen dik üçgen tanımına göre:

- x > 0 [xPozitif]
- y > 0 [yPozitif]
- z > 0 [zPozitif]
- $z = \sqrt{(x^2 + y^2)}$ [pisagor]
- (x+y) > z & |x-y| < z & (x+z) > y & |x-z| < y & (z+y) > x & |z-y| < x [üçgen]

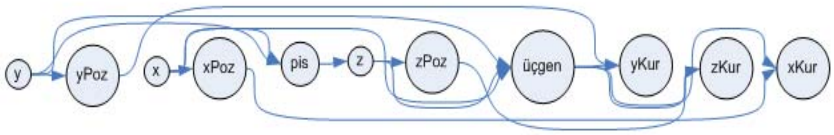
kuralları belirlenebilir. Bu kurallar çerçevesinde geçerlemeler:

- x.Geçerli = xPozitif & üçgen [xKural]
- y.Geçerli = yPozitif & üçgen [yKural]
- z.Geçerli = zPozitif & üçgen [zKural]

olarak belirlenebilir.



Şekil 6. Örnek Çizge Yapısı (ebeveyn çocuk hiyerarşisi)



Şekil 7. Sıralanmış çizge

Şekil 7'de çizgenin örnek bir sıralanması verilmiştir. Bu sıralama üzerinden gidilecek olursa y değerinde herhangi bir değişme olduğunda zKur düğümüne kadar düğümler kontrol edilecek ve bağımlı olan deyimler işlenecektir. x değerinde değişme olduğunda ise xKur'a kadar olan düğümler incelenecektir.

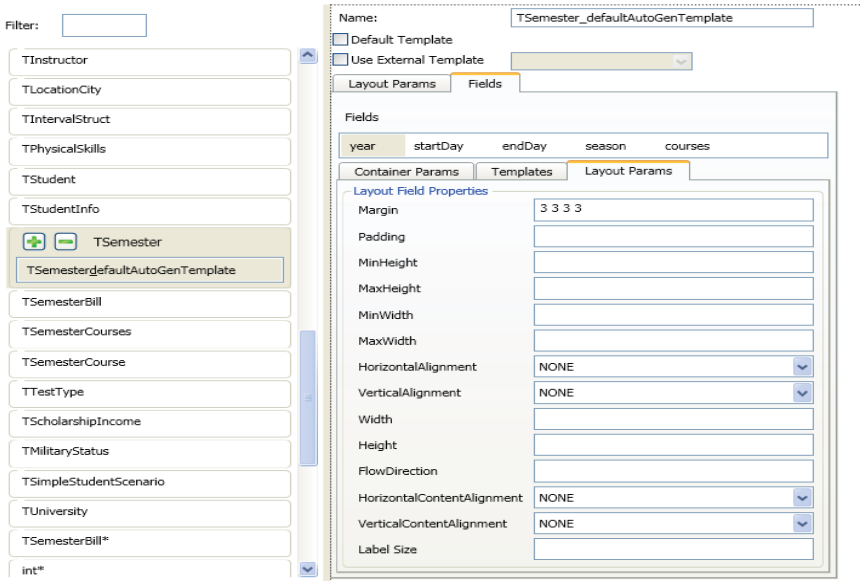
Çizge sırası oluşturulurken topolojik sıralama yapılmış ve öyle oluşturulmuştur. Bu çizge kullanarak sadece ilgili bağlantılar yeniden kontrol edildiğinden ve gereksiz kontroller azaltıldığından sistemin hızlanması sağlanmıştır.

Bir başka gelişim ise geçerleme kurallarının işlenmesinde yapılmıştır. KABAN-1 sisteminde geçerleme kuralları XML dosyası şeklinde tutulmakta ve serileştirme işlemi yapılmaktadır. İhtiyaç halinde "Reflection" kullanılarak veri ağacı gezilmekteydi. KABAN-2 sisteminde geçerleme kuralları KVF dosyalarında tutulmakta ve sistem ayağa kalkmadan otomatik bir uygulamamız sayesinde C# kodlarına dönüşmektedir. Bu sebeple geçerleme işlenirken "Reflection" kütüphanesi yerine direk kod blokları kullanılmaktadır. Yaptığımız incelemelerde sadece bu işlemin sistemi yaklaşık 5-10 kat hızlandırdığı gözlemlenmiştir.

3.2.3 Konfigürasyon Dosyaları

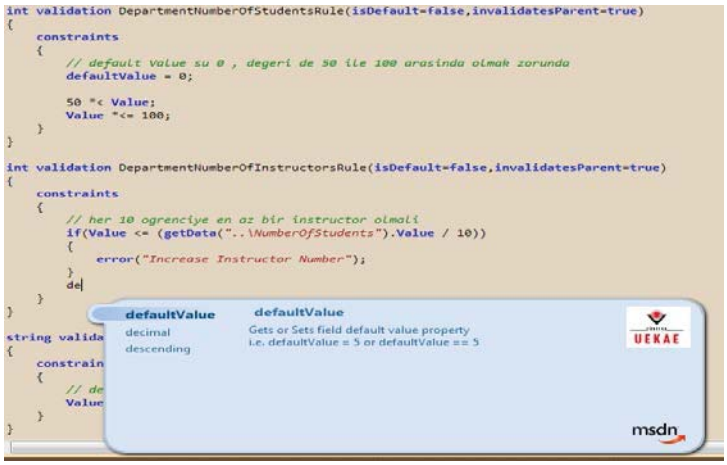
KABAN-1 sisteminde yapılandırma dosyaları kullanıcının rahatlıkla değiştirebilmesi ve bu değişikliklerin çok işgücü çıkarmaması adına XML tabanlı tanımlanmıştır. Her ne kadar XML dosyaları ile uğraşmak yazılımcılar için kolay olsa da diğer disiplinlerden kullanıcılar için alışma süreci o kadar kolay olmamaktadır. Bu sebeple bu dosyalar daha kolay nasıl oluşturabileceğinin yöntemleri araştırılmış ve alternatif çözüm bulunmuştur. Editör yapılandırma dosyasının yerine bir uygulama kullanılmasına karar verilmiştir. Bu uygulama Xaml kodu üretmektedir ve KABAN-2 sistemi bu Xaml kodlarını kullanarak arayüz üretmektedir. Şema dosyasından veri yapıları okunduktan sonra bu uygulama açılmakta ve her veri tipi için kullanıcıya ilgili kutucuğa kullanıcı isteğini girmesine imkân vermektedir.

Şekil 8 de bu uygulamanın demo projesi için arayüzü görülmektedir. Bu vesileyle editör tarafı için yapılandırma dosyası oluşturulması işlemi kolaylaştırılmıştır.



Şekil 8. KABAN-2 Arayüz Editörü

Bir başka yapılandırma dosyası da geliştirici tarafından ilgili geçerlemeleri tanımlamak için kullanılmaktadır. Buna çözüm olarak yeni bir betik(script) dili geliştirilmiş ve bu betik dili için derleyici tanımlanmıştır. Yeni dil mümkün olduğunca herkesin anlayabileceği kadar basit bir şekilde tanımlanmaya çalışılmış ve en sık kullanılan geçerleme tiplerini en sade ve en hızlı yazımını kolaylaştıracak şekilde tasarlanması gayesi güdülmüştür(Şekil 9). Yazım kolaylığını sağlamak adına yardımcı(Intellisense) özelliği eklenmiştir. Kullanıcının yazdığı kelimeler otomatik olarak tamamlanmaktadır.



Şekil 9. KABAN-2 Geçerleme Dili

```
syntax Rule = caption:RuleCaption body:RuleBody => Rule{caption,body};

syntax RuleCaption =  valt:Identifier vall:ValidationToken rulnm:Identifier opp:OpenPar  IsDefaultToken AssignmentOperator logv1:LogicalValue comm:CommaToken
InvalidatesParentToken AssignmentOperator logv2:LogicalValue clp:ClosePar  => Rule_Caption{Val_Type{valt}, Rule_Name{rulnm}, Is_Default{logv1}, Invalidates_Par

syntax RuleBody =  OpenCurlyBracket innb:RuleMainBody CloseCurlyBracket => Rule_Body{innb};

syntax RuleMainBody =  ch:ChildValidations => Rule_Main_Body{No_Tag{ch}}
                     | ty:TypeValidationContent => Rule_Main_Body{No_Tag{ty}}
                     | ch:ChildValidations ty:TypeValidationContent => Rule_Main_Body{No_Tag{ty} , No_Tag{ch}}
                     | ty:TypeValidationContent ch:ChildValidations => Rule_Main_Body{No_Tag{ty} , No_Tag{ch}};



syntax ChildValidations = FieldsToken OpenCurlyBracket block:MemberValidation+ CloseCurlyBracket => ChildValidations{valuesof(block)};

syntax MemberValidation = id:Identifier opr:AssignmentOperator val:StringConstant SemicolonToken => MemberValidation{Field_Name{id},Operator{opr},Field_Value{v


syntax TypeValidationContent = sc:SingleConstraintOperation => No_Tag{sc}
                             | sf:SingleFunctionOperation => No_Tag{sf}
                             | sv:SingleVariableOperation => No_Tag{sv}

                             | sc:SingleConstraintOperation sf:SingleFunctionOperation+ => No_Tag{sc,sf}
                             | sf:SingleFunctionOperation+ sc:SingleConstraintOperation  => No_Tag{sc,sf}

                             | sc:SingleConstraintOperation sv:SingleVariableOperation+ => No_Tag{sc,sv}
                             | sv:SingleVariableOperation+ sc:SingleConstraintOperation  => No_Tag{sc,sv}

                             | sf:SingleFunctionOperation+ sv:SingleVariableOperation+ => No_Tag{sf,sv}
                             | sv:SingleVariableOperation+ sf:SingleFunctionOperation+ => No_Tag{sf,sv}

                             | sf:SingleFunctionOperation+ sc:SingleConstraintOperation sv:SingleVariableOperation+ => No_Tag{sc,sf,sv}
                             | sf:SingleFunctionOperation+ sv:SingleVariableOperation+ sc:SingleConstraintOperation => No_Tag{sc,sf,sv}
                             | sc:SingleConstraintOperation sf:SingleFunctionOperation+ sv:SingleVariableOperation+ => No_Tag{sc,sf,sv}
                             | sc:SingleConstraintOperation sv:SingleVariableOperation+ sf:SingleFunctionOperation+ => No_Tag{sc,sf,sv}
                             | sv:SingleVariableOperation+ sc:SingleConstraintOperation sf:SingleFunctionOperation+ => No_Tag{sc,sf,sv}
                             | sv:SingleVariableOperation+ sf:SingleFunctionOperation+ sc:SingleConstraintOperation => No_Tag{sc,sf,sv};

syntax SingleVariableOperation = exa:ExtendedVarAssignment SemicolonToken => Rule_Variable_Assignment{exa};

syntax SingleConstraintOperation = ConstraintsToken OpenCurlyBracket block:ConstraintStatement* CloseCurlyBracket => Constraint_Block{valuesof(block)};
```

Şekil 10. KABAN-2 Geçerleme Dili BNF tanımı

Şekil 10 da yeni geliştirilen dilin BNF gösteriminde tanımının bir kısmı görülmektedir. Derleyici bu tanıma bakarak dosyanın doğruluğunu ve derlenebildiğini kontrol etmektedir. Eğer dosyada bir hata yoksa dosya derlenip C# koduna dönüşmektedir.

4 SONUÇ

KABAN-1 ve KABAN-2 sistemleri benzetim projeleri gibi hem veri girdisinin çok ve karmaşık olduğu hem de veri bütünlüğünün ve doğruluğunun önem arz ettiği projelerde olası iş yükünü en aza indirme çabası ile üretilmiş altyapı projeleridir. KABAN-1 projesi TÜBİTAK-İLTAREN bünyesinde çok sayıda projede kullanılmış ve güvenilirliğini çok kez kanıtlamış bir projedir. KABAN-1 projesinde iyileştirme önerileri incelenmiş ve KABAN-2 projesi geliştirilmiştir. Bu vesileyle hem sistem hızlandırılmış hem de kullanım kolaylığı artmıştır. KABAN projeleri özünde birbirinden bağımsız bir geçerleme ve editör oluşturma altyapısını barındırmaktadır. İsteğe göre sadece geçerleme kısmı tek başına kullanılabilmektedir.

Müşteri istekleri zamanla değişen, gelişen uzun soluklu projeler göstermiştir ki KABAN kullanımı projelerin geliştirme maliyetinde ve zamanında kazanç sağlamıştır. Her altyapı projesinde olduğu gibi ilk kullanımında bir öğrenme maliyetine gereksinim duyulmasına rağmen sonuçta büyük resme bakıldığında projelerin daha hızlı tamamlanmasını sağladığı kurumumuzda edinilmiş bir tecrübedir.

5 KAYNAKLAR

[1] Demir I, Tezcan Z, Alpdemir M.N, KABAN : Simulasyon Verileri Geçerleme ve Otomatik Kullanıcı Arabirimi Oluşturma Altyapısı , USMOS, 2007

[2] Commons Validator, http://jakarta.apacheçorg/commons/validator, Apache Software Foundation, 2006

[3] DataSift: Data Validation and Transformation Framework, http://www.datasift.org, The Auel Project, 2004

[4] Shallman, D., i-screeen: The Java Object Validation Framework, http://i-screen.org/, 2006

[5] Balcı, O.,"Verification, Validation, and Testing",in Handbook of Simulation, Eds. Jerry Banks, John Wiley & Sons, 1998

[6] Petrakos, G.A. and Farmakis, G.E., "A Declarative Approach To Data Validation Of Statistical Data Sets, Based On Metadata", http://citeseer.ist.psu.edu/ petrakos00declarative.html, 2000

[7] XML Schema Part 0: Primer, W3C Recommendation, Second Edition, http://www.w3.org/XML/Schema, 28 October 2004

[8] Dossot David, NxBRE .Net Business Rules Engine, 2006

[9] Hanson Eric N.,Rule Condition Testing and Action Execution in Ariel,In Proceeding of ACM SIGMOD Conference, 1992

[10] Hanson Eric N.,The Design and Implementation of the Ariel Active Database Rule System, IEEE Transactions on Knowledge and Data Engineering archieve, Volume 8 Issue1, Pages 157-172,1996

[11] Widow Jennifer, The Starbust Active Database Rule System, IEEE Transactions on Knowledge and Data Enginnering, 1996

[12] Cho Eunsuk, Park Insuk, Hyun Soon J.,Kim Myungchul, ARML: an Active Rule Markup Language for Sharing Rules among Active Information Management Systems, First International Workshop on RuleML, 2002

[13] Marconi Micheal, Nentwich Christian, CLIX Langıage Specification Version 1.0, 2004

[14] Boley Harold, Tabet Said, Wagner Gerd, Design Rationalite of RuleML: A Markup Language for Semantic Web Rules, Semantic Web Working Symposium, 2001

[15] Boley Harold, Tabet Said, Wagner Gerd, Design Rationalite of MOF-RuleML: The Abstract Syntax of RuleML as a MOF Model, OMG Meeting, 2003