Cobalt: Test Uygulamaları için Protokol Kütüphanesi

Uğur ZÖNGÜR ve S. Tuncer ERDOĞAN

REHİS SMD Test Mühendisliği Müdürlüğü,
Aselsan A.Ş., Ankara, Türkiye
{uzongur,terdogan}@aselsan.com.tr
http://www.aselsan.com.tr

Özet Yazılım geliştirme, test ve kabul safhalarında, test altındaki yazılım birimlerinin haberleştiği gerçek birimlerin benzetimini yapabilen test uygulamalarına ihtiyaç duyulmaktadır. Test uygulaması geliştirme sürecindeki tekrarlanan örüntüleri ortaklayarak, harcanan mühendislik uğraşını azaltmak üzere geliştirilen Cobalt, öntanımlı haberleşme protokollerindeki karmaşık veri tipleri için, genişletilebilir bir görüntüleme, düzenleme, serileştirme ve ayrıştırma kütüphanesidir. Kütüphanenin temel kullanım alanı, üçüncü taraflarca tanımlanmış haberleşme protokollerini kullanan yazılım birimlerini test etmek için gerekli altyapıyı sunmaktır. Bu makalede, Cobalt kütüphanesinin geliştirilmesinin ardındaki motivasyon, gerçeklenmesine ilişkin teknik detaylar ve kütüphanenin kullanımı ile elde edilen faydalar anlatılmaktadır.

Anahtar Kelimeler Yazılım test, haberleşme protokolleri, serileştirme, ayrıştırma, test uygulamaları, test bileşenleri, simülatör

1 Giriş

Yazılım ve donanım birimlerinin testlerinde, izole edilmiş bir test ortamı oluşturabilmek için, birimlerin veri alışverişi sağladığı çevre bileşenleri taklit eden bir uygulama kümesinin (Test uygulamaları/Simülatörler) hazırlanması gerekmektedir. Gerçek birimlerin testler esnasında kullanılmasının maliyetli/imkansız olduğu durumlarda ya da test vektörlerinin kontrollü bir biçimde test altındaki birime beslenmesi, bu birimden alınan çıktıların elde edilmesi ve analizinde, test uygulamaları önemli bir rol oynamaktadır. Test uygulamalarından beklenen temel özellikler, hızlı prototipleme ve *anlık*[1] testler için kullanıcı arayüzüne sahip olması ve otomatik testlerin yürütülmesi için bir uygulama programlama arayüzü[2] (API) sunmasıdır.

Test uygulamaları, yazılım ve sistem geliştirme süreçleri boyunca, şirket içerisinde test, yazılım, sistem mühendislikleri, üretim gibi bir çok bölüm tarafından kullanılmaktadır. Buna ek olarak, bu uygulamalardan, son kullanıcının dahil olduğu kabul muayene testlerinde de yararlanılabilmektedir. Bahsedilen

[1] Ad-hoc
[2] Application Programming Interface - API

test aşamalarında yer alan bu uygulamaların, test edilecek tüm birimlerin her arayüzü için oluşturulması ve idame edilmesi gerekmektedir. Bu sebeple bir çok birimden oluşan büyük çaplı projelerde yüksek mühendislik maliyetleri ortaya çıkmaktadır.

Cobalt kütüphanesinin geliştirilmesindeki amaç, test uygulaması geliştirme sürecindeki tekrarlanan örüntüleri ortaklayarak, harcanan mühendislik eforunu azaltmak ve söz konusu maliyetleri düşürmektir. Cobalt, öntanımlı haberleşme protokollerindeki karmaşık *ikili*[3] veri tipleri için, genişletilebilir bir yapıya sahip görüntüleme, düzenleme, serileştirme ve ayrıştırma kütüphanesidir.

Cobalt bir takım modern uygulama geliştirme kütüphaneleriyle paralel olarak, "kendini tekrar etme"[4] [1] ve "genel geçer kurallar konfigürasyondan önce gelir"[5] prensipleri çerçevesinde geliştirilmiştir.

Kendini Tekrar Etme (DRY). Seri veri üretimi, ayrıştırılması, grafik arayüz üzerinde verilerin gösterimi ve düzenlenmesi işlevlerinin davranışları, sadece protokol tanımına bağlı olarak değişmektedir. Cobalt kütüphanesi bu işlevlerin gerçeklenmesini protokol tanımından çıkarım yaparak geliştiricinin üzerinden almakta ve bir bilginin sadece bir kez ve en ilgili olduğu yerde bulunmasını sağlamaktadır.

Genel Geçer Kurallar Konfigürasyondan Önce Gelir. Cobalt, kütüphane genelinde geliştiriciye, varsayılan değer ve davranışları gerçekleyerek sunmaktadır. Bu sayede genel istekler çerçevesinde geliştirilecek test uygulamaları daha hızlı ve hatasız biçimde oluşturulabilirken, genişletilebilir yapısı sayesinde özel durumlar da desteklenmektedir.

Bu makalede Cobalt kütüphanesinin gerçeklenmesi ve kazanımlarına ilişkin bilgi ve değerlendirmeler aktarılmıştır. Bölüm 2'de Cobalt'la benzer görevleri yerine getiren çalışmalardan bahsedilmiştir. Bölüm 3'te Cobalt kütüphanesinin teknik altyapısı, mimari tercihleri ve nedenleri anlatılmış, Bölüm 4'te Cobalt'ın kullanımı ile elde edilen kazanımlara yer verilmiştir. Bölüm 5 ise sonuç bölümüdür.

2 Benzer Çalışmalar

Bugüne kadar çok sayıda *ikili* veri serileştirme/ayrıştırma teknolojisi geliştirilmiştir. Bunlar arasında bulunan Protocol Buffers[2], Thrift[3], CORBA[4], ASN.1 (BER, CER, DER)[5] gibi teknolojiler, olgunlaşmış ve kendilerine yaygın kullanım alanı bulmuşlardır. Ancak, haberleşme protokollerindeki kodlanmış veri biçimlerini kendileri tanımladıkları için, özelleşmiş kodlamaya sahip *ikili* verileri

[3] Binary
[4] Don't Repeat Yourself - DRY
[5] Convention over configuration

ifade edemezler. Hedefi, *eski*[6] veya performans odaklı, özelleşmiş protokolleri gerçeklemek olan Cobalt ile bu noktada ayrılmaktadırlar.

Az sayıda olmasına karşın, özelleşmiş protokolleri destekleyebilen teknolojiler de bulunmaktadır. Ragel[6] gibi araçlar, serileştirme/ayrıştırma yeteneğini kod üreterek elde ederken, construct[7] ve protlib[8] gibi kütüphaneler, Cobalt'a benzer biçimde, bu yeteneği çalışma zamanında gerçekleştirmeyi tercih etmiştir. Ancak bu teknolojiler verileri grafiksel kullanıcı arayüzünde gösterme yeteneğine sahip olmadığından, test uygulaması geliştirme sürecinde istenen faydayı sağlayamamaktadır.

3 Cobalt

Cobalt kütüphanesi, *ikili* verileri modellemek için bir ağaç yapısını temel almaktadır. Bu ağaç yapısı, *ikili* verilerin *ayrıştırma ağaçlarının*[7] bir taslağı/prototipi niteliğindedir. Kütüphane aynı yapı üzerinden hem serileştirme, hem ayrıştırma, hem de kullanıcı arayüzüne ilişkin gösterim ve düzenleme işlevlerini yerine getirmektedir. Bu bölümde, bu işlevlere ait teknik detaylara yer verilmiştir. Bölüm 3.1'de veri modeli açıklanmıştır. Bölüm 3.2'de seri veri üretimi ve ayrıştırılmasından bahsedilirken, Bölüm 3.3'te grafik arayüz gerçeklemesi anlatılmıştır. Bölüm 3.4'te ise üçüncü taraflarca geliştirilmiş uzaktan metod çağırma benzeri teknolojiler ile Cobalt'ı entegre edebilmek için tasarlanan POJO/POJI eşleyiciye yer verilmektedir.

3.1 Veri Modeli

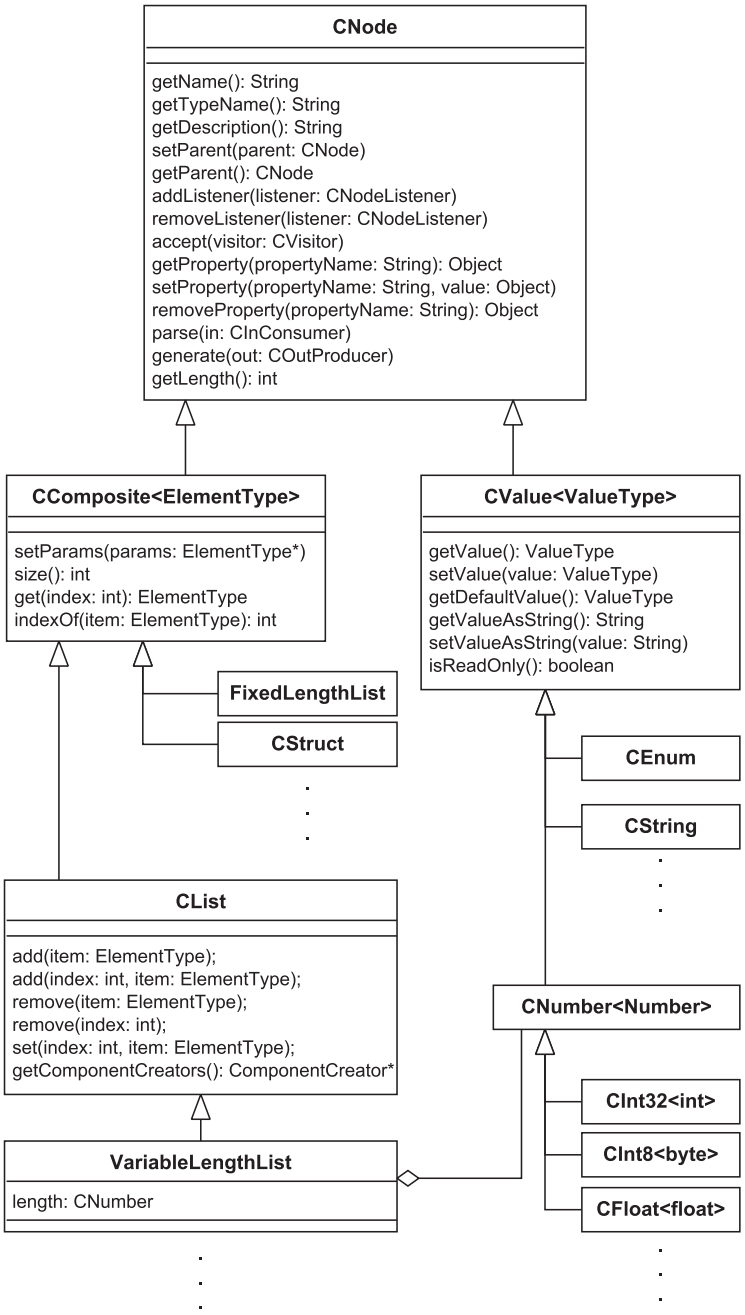
Cobalt ile geliştirilen test uygulamalarında, üzerinde işlem yürütülen verilerin `CNode` adı verilen düğümlerden oluşan sıralı bir ağaç yapısı şeklinde ifade edilmesi benimsenmiştir. Yaprakları temsil eden `CValue` ve başka `CNode` düğümlerini içerebilen `CComposite` yapısı, bu `CNode` tipinden türetilmiştir. `CValue` düğümleri genelde tam sayı, kayan noktalı sayı, karakter gibi ilkel ya da karakter dizisi, çoklu seçim alanı gibi daha karmaşık veri tipleri olarak kendini göstermektedir. `CComposite` yapısı ise sabit ve değişken uzunlukta listeler, veri desteleri gibi yapılara karşılık gelmektedir.

Her düğüm isim, tip ismi, açıklama gibi meta-bilgileri temin etmenin yanı sıra ebeveyn düğümün atanması/elde edilmesi, dinleyici eklenmesi/çıkarılması gibi işlevleri de yerine getirmektedir.

Cobalt'ın yapısına dair genel bir fikir oluşturmak adına, kütüphaneye ait UML sınıf diyagramı Şekil 1'de verilmiştir. Bu şekil, Cobalt bünyesindeki tüm sınıfları, sınıf metodlarını ve sınıf üye değişkenlerini kapsamamakta, bunların örnek bir alt kümesini sunmaktadır.

Kütüphanenin içerisinde ön tanımlı bir çok veri tipi bulunmakla birlikte, ihtiyaç duyulduğu takdirde yeni veri tipleri, kütüphane içerisinde bulunan soyut

[6] Legacy
[7] Parse tree



Şekil 1. Cobalt kütüphanesi sınıflarının örnek bir alt kümesinin diyagramı

sınıflar kullanılarak genişletilebilmektedir. Bu genişleme noktaları Bölüm 3.2 ve 3.3'de bahsedilen seri veri üretimi/ayrıştırılması ve görüntüleme/düzenleme davranışlarının belirlenmesinde geliştiricilere esneklik sağlamaktadır.

3.2 Seri Veri Üretimi ve Ayrıştırılması

Tüm haberleşme kanalları, seri haldeki veri ile çalışmakta olduğundan, Cobalt veri ağacının serileştirilmesi gerekmektedir. Cobalt kütüphanesi geliştirilmeden önce, test uygulamalarında kullanılan verilerin uygulama içerisindeki gösterimleri olan modellerin haberleşme kanalına uygun hale getirilmesi işlemi, modele özelleştirilmiş kodların hazırlanmasını gerektirmekteydi. Bu yöntem, yukarıda bahsedilen DRY prensibi ile çelişerek, protokol üzerinde yapılan değişikliklerin hem modelde, hem de serileştirme/ayrıştırma mekanizmasında güncellenmesi ihtiyacını doğurmaktaydı. Bahsedilen işlemin otomatik olarak modelden türetilerek yürütülmesi, geliştirme/idame sürecini kısaltırken, geliştirici kaynaklı hataları da azaltan sağlıklı bir çözümdür.

Cobalt kütüphanesinde, düğümlerin kendi seri veri üretimi ve ayrıştırmasından sorumlu olduğu bir mimari benimsenmiştir. Yaprak düğümleri sadece taşıdığı veriyi serileştirirken, birleşik yapılar bu işlemi özyineli olarak çocuklarına delege etmektedir. Bu sayede, kök düğüm üzerinde yapılan serileştirme/ayrıştırma isteği tüm ağaç yapısına yayılmakta ve düğümler tarafından haberleşme kanalına doğru sıra ile iletilmektedir.

Fiziksel Haberleşme Protokolü Bağımsızlığı. Veri ağacı üzerindeki düğümler, serileştirme ve ayrıştırma işlemleri için, Cobalt kütüphanesi içerisindeki `COutProducer` ve `CInConsumer` arayüzlerini gerçekleyen sınıflara ihtiyaç duymaktadır. Bu sınıflar haberleşme katmanının soyutlaması olup, veri modelinin serileştirme ve ayrıştırma işlemini, fiziksel haberleşme protokolünden[8] ve bu protokolün *bayt sıralaması*[9] gibi ayrıntılarından bağımsız hale getirmektedir. Java platformunda haberleşme işlemleri için fiili standart olarak kullanılan `InputStream` ve `OutputStream` arayüzlerini saran sınıflar kütüphane içerisinde sağlandığından, varolan fiziksel haberleşme katmanları kolayca entegre edilebilmektedir.

Serileştirme/Ayrıştırma Genişletilebilirliği. Cobalt kütüphanesi içerisinde geniş bir öntanımlı veri tipi seti bulunmakla birlikte, öngörülemeyen serileştirme/ayrıştırma ihtiyaçlarının karşılanabilmesi için bazı yöntemler sunulmaktadır.

Bu yöntemlerden en çok kullanılanı, `CAbstractComposite` soyut sınıfı temel alınarak birleşik veri tiplerinin oluşturulmasıdır. Bu yöntem esasen Cobalt ile veri tanımlamanın bel kemiğini oluşturmakta, iç içe uygulandığında karmaşık veri tiplerinin ifade edilmesine olanak tanımaktadır.

[8] Ağ haberleşmesi, dosya okuma/yazma, seri kanal, vb.
[9] Endianness

Yöntemlerden bir diğeri ise, yeni ilkel tip ihtiyaçlarını hedefler ve Cobalt ilkel tip sınıflarından birinin kalıt alınması ile gerçekleştirilir. İstenen ihtiyaca en yakın sınıfa ait `parse` ve `generate` metodlarının varsayılan davranışları değiştirilerek ya da `CValue` soyut arayüzünün gerçeklenmesi suretiyle, istenen veri tipinin elde edilmesine olanak tanınmaktadır. Örneğin, değişken uzunlukta kodlamalı tam sayılar[10][9] Cobalt kütüphanesi içerisinde yer almamakta, ancak kolayca eklenebilmektedir. Diğer bir örnek olarak, bit duyarlılığında veri yapıları Cobalt'ın ilk versiyonlarında desteklenmemesine karşın, doğan ihtiyaçlar doğrultusunda kütüphane genişletilerek gerçeklenebilmiş ve daha sonraki versiyonlarda kod tabanına dahil edilmiştir.

Ziyaretçi Örüntüsü. Tüm ağaç yapısının harici/merkezi bir algoritma ile dolaşılması ihtiyacı göz önünde bulundurularak Cobalt kütüphanesinde düğümlere ziyaretçi örüntüsü desteği eklenmiştir. Böylelikle var olan sınıflar ve davranışları değiştirilmeden, ağaç üzerinde gerçekleştirilecek yeni bir işlemin tanımlanabilmesi sağlanmıştır.

Örnek olarak, C/C++ içerisinde tanımlanan `struct` ve `class`'ların hafıza içerisindeki yerleşimlerinde, platforma ve derleyiciye bağlı olarak anlamlı veri alanları dışında hafıza bölgeleri eklenmektedir[11]. Eklenen bölgelerin hesaplanması için model ağacı üzerinde serileştirme ve ayrıştırmadan önce analiz yapılması ve veri tiplerinden hizalama bilgilerinin alınması gereklidir. Cobalt'ın ilk versiyonlarında bu durum öngörülememesine karşın, ziyaretçi örüntüsü kullanılarak bahsedilen yetenek gerçeklenebilmiş ve daha sonra da Cobalt kod tabanına eklenmiştir.

Ziyaretçi örüntüsü yukarıda belirtilen analiz işlemleri için kullanılabileceği gibi serileştirme/ayrıştırma yeteneğinin tamamen farklı bir biçimde yönetilmesi amacıyla da kullanılabilir. Örneğin bir düğümün XML veya JSON formatına aktarılması ve bu formattan okunması kolaylaştırılmıştır.

3.3 Grafiksel Kullanıcı Arayüzü

Yazılım geliştirme/idame sürecinde en fazla zaman ve efor gerektiren mühendislik kalemlerinden biri de grafiksel kullanıcı arayüzü geliştirme safhasıdır [10]. Bu durum elle geliştirilen/idame edilen kullanıcı arayüzüne sahip test uygulamaları için de geçerlidir. Bu şekilde geliştirilen test uygulamalarında, gönderilecek/alınacak verilere ait her bir alanın düzenlenmesi ve görüntülenmesi için ayrı ayrı arayüz elemanlarının oluşturulması gerekmektedir. Listeler ve iç içe geçmiş, yüksek seviyede hiyerarşiye sahip veriler gibi karmaşık yapılar göz önüne alındığında bu durumun maliyete etkisi daha iyi anlaşılmaktadır.

Bölüm 3.2'ye benzer biçimde, grafiksel kullanıcı arayüzü de haberleşme protokolüne bağımlıdır. Elle hazırlanan/idame edilen grafiksel kullanıcı arayüzleri için, protokolde yapılan herhangi bir değişiklik, hem model hem de kullanıcı arayüzünün mükerrer güncellemelerini zorunlu kılmaktadır. Bu durum, geliştirme

[10] Google, Protocol Buffers, varint gerçeklemesi
[11] Padding

maliyetlerini arttırmakla kalmayıp, geliştirici kaynaklı hataların da çoğalmasına sebep olmaktadır. Ayrıca son kullanıcıya teslim edilmeyen test uygulamalarında, hızlı geliştirme adına, kötü kodlama alışkanlıkları ve anti-örüntülere göz yumulabilmekte ve sonuç olarak geniş kod tabanına sahip olabilen test yazılımlarının idamesi zorlaşmaktadır.

Cobalt'ta, haberleşme protokolünü tanımlayan veri modelinin çalışma zamanında analizi ile kullanıcı arayüzü otomatik olarak oluşturulmaktadır. Kullanıcı arayüzünün geliştirici veya kullanıcı müdahalesine ihtiyaç duyulmadan ortaya çıkması, DRY prensibi çerçevesinde, kodun değişen kısmını protokolün tanımlandığı yere hapsetmekte, model-grafik arayüzü arasındaki bağlaşımı ortadan kaldırmaktadır. Böylece, yukarıda bahsi geçen olumsuzlukların önüne geçilmektedir.

Cobalt veri ağacının görüntüleme ve düzenleme işlevleri, kütüphane içerisinde bulunan `CNodeExplorer` bileşeni kullanılarak yapılabilmektedir. Bir *ağaç-tablo*[12] gerçeklemesi olan `CNodeExplorer`, verinin ağaç yapısını birebir yansıtmaktadır. Düğümlere ait veri ve meta-verilerin görüntülenmesini ve ilgili alanların düzenlenmesini sağlayan bu bileşen, kullanım ve görünüm açısından modern bütünleşik geliştirme ortamlarında bulunan hata ayıklama pencerelerini anımsatmaktadır.

`CNodeExplorer` bileşeninin sağladığı bir diğer avantaj da, geliştirilen test uygulamalarının aralarında *birörnek*[13] olmasıdır. Böylelikle görüntüleme ve düzenleme ekranlarının geliştiriciden geliştiriciye büyük farklılık göstermesinin önüne geçilmiş, kullanıcı adaptasyonu kolaylaştırılmıştır.

Kullanıcı Arayüzü Genişletilebilirliği. Cobalt'ın çekirdek yeteneklerinin dışında bir görüntüleyici veya düzenleyici ihtiyacı doğduğunda, istenen herhangi bir veri tipi için özelleşmiş *yerinde*[14] arayüz bileşenleri eklenebilmektedir. Örneğin renk verisi taşıyan bir `CNode`'un yerinde bir renk editörü ya da bir renk dialogu ile düzenlenmesi sağlanabilmektedir.

3.4 POJO/POJI Eşleyici

Cobalt, Bölüm 3.2'de bahsedildiği biçimde özelleştirilmiş protokoller için serileştirme/ayrıştırma yapabilmesinin dışında, bu yeteneğin hali hazırda gerçeklendiği veya buna ihtiyaç duyulmayan bazı teknolojiler için de görüntüleyici ve düzenleyici olarak kullanılabilmektedir. CORBA, RMI, Thrift, OSGi gibi uzaktan metod çağırma/servis yönelimli mimari ara katman teknolojileri, haberleşme soyutlaması olarak Java arayüzlerini ya direkt, ya da bir *arayüz tanımlama dili*[15] vasıtasıyla üreterek kullanmaktadır. Cobalt, bu arayüzlerin hangi protokolü kullandığına bakmaksızın, platformun sağladığı *tip içgözlemi*[16] ve *yansıtma*[17] yetenekleri ile bu arayüzleri ve bahsedilen arayüzlerin metod argümanları ve dönüş

[12] TreeTable
[13] Uniform
[14] In-Place
[15] Interface Description Language - IDL
[16] Type introspection
[17] Reflection

parametrelerine ait objeleri analiz ederek görüntüleyebilmekte ve düzenleyebilmektedir.

Herhangi bir Java konvansiyonunu takip etmeyen arayüz ve nesneler, yalın eski Java arayüzü (Plain Old Java Interface - POJI) ve yalın eski Java nesnesi (Plain Old Java Object - POJO) olarak adlandırılmaktadır. Cobalt bu arayüz ve nesneler üzerinde herhangi bir kısıtlamayı zorunlu tutmadığından ve bu obje ve arayüzleri daha önce bahsedilen `CValue` ve `CComposite` düğümlerine eşlediğinden bu yetenek POJO/POJI eşleyici olarak adlandırılmıştır.

POJO/POJI eşleyici, karşı taraftan çağırılan metodları analiz edebilmek için, çalışma zamanında, ilgili servis arayüzünün gerçeklemesi olan bir vekil nesnesi oluşturmaktadır. Bu vekil nesne kendisine yapılan metod çağrılarında *kütük tutma*[18] veya kendisini dinleyen sınıfları uyarma işlemini yürütmektedir. Bu yöntem sadece "somut olmayan" arayüzlere uygulanabilir ancak uzaktan metod çağırma/servis yönelimli mimari teknolojileri, karşı tarafa ait, çağırılacak metodlar için "somut" bir vekil nesne oluşturmaktadır. Dolayısıyla, karşı taraftan çağırılan metodlar için izlenen yöntem bu durumda izlenememektedir, fakat yine de vekil nesneye yapılan metod çağrılarında araya girmek gerekmektedir. Aslında bir *bağımlılık enjeksiyonu*[19] kütüphanesi olan Guice[11], *çalışma/yükleme zamanı dikişi*[20] ile *ilgiye yönelik programlama*[21] da yapabildiği için bahsedilen araya girme işlemini gerçekleştirebilmektedir. Yukarıda bahsedilen iki yöntem kullanılarak, bir *sarmalayıcı*[22] vasıtasıyla tekörnek biçimde, gönderilen ve alınan mesajlar (karşı taraftan çağırılan ve karşı tarafta çağırılacak metodlar) için istenen davranış kolayca belirlenebilmektedir.

POJO/POJI eşleyici, eşleme işlemini otomatik olarak yaptığından dolayı, çağırılan veya çağırdığı metodların değişmesi durumunda, herhangi bir geliştirici veya kullanıcı müdahalesine gerek duymamaktadır.

3.5 Örnek Kullanım

Cobalt kullanılarak tanımlanmış bir veri tipinin kaynak kodları aşağıda verilmiştir.

```
public class AracTipi extends CEnum {
    public static final Int8 KARA  = new Int8("Kara",  0);
    public static final Int8 DENIZ = new Int8("Deniz", 1);
    public static final Int8 HAVA  = new Int8("Hava",  2);

    public AracTipi() {
        super("Araç tipi", KARA, DENIZ, HAVA);
    }
}
```

[18] Logging
[19] Dependency Injection - DI
[20] Runtime/Loadtime Weaving
[21] Aspect Oriented Programming- AOP
[22] Wrapper

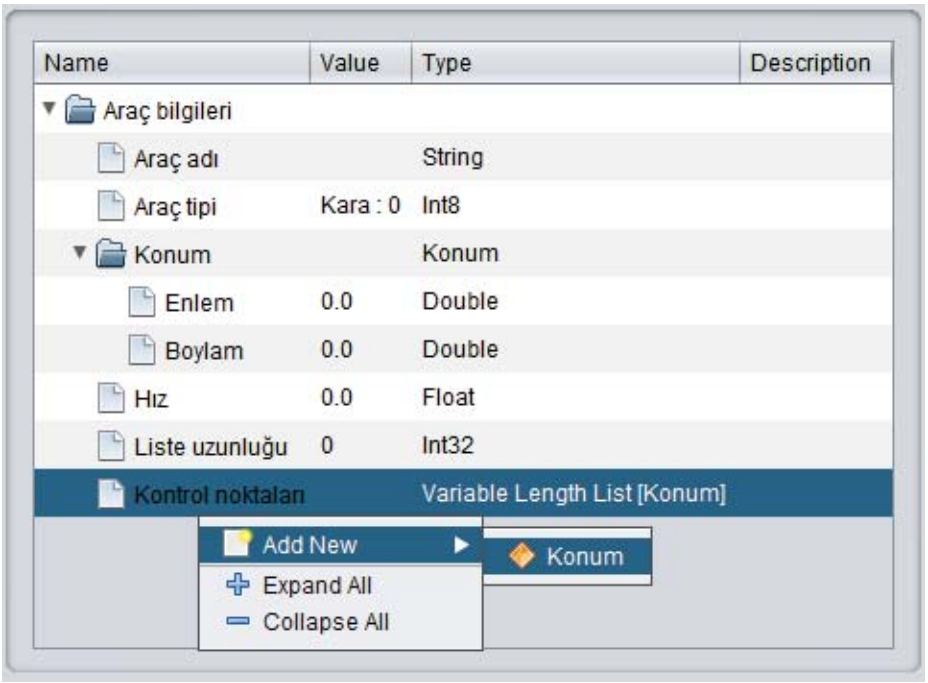
```
public class Konum extends CAbstractComposite {
    protected CDouble enlem  = new CDouble("Enlem");
    protected CDouble boylam = new CDouble("Boylam");

    public Konum() {
        setParams(enlem, boylam);
    }
}

public class AracBilgileri extends CAbstractComposite {
    protected NullTerminationString aracAdi =
        new NullTerminationString("Araç adı");
    protected AracTipi aracTipi = new AracTipi();
    protected Konum konum = new Konum();
    protected CFloat hız = new CFloat("Hız");
    protected Int32 listeUzunlugu = new Int32("Liste uzunluğu");
    protected VariableLengthList<Konum> kntrlNoktalari =
        new VariableLengthList<Konum>
            ("Kontrol noktaları", listeUzunlugu, Konum.class);

    public AracBilgileri() {
        setParams(aracAdi, aracTipi, konum, hız,
            listeUzunlugu, kntrlNoktalari);
    }
}
```

Bu veri tipinin, `CNodeExplorer` bileşenindeki gösterimi Şekil 2'de sunulmaktadır. Bu grafiksel kullanıcı arayüzü bileşeni, Bölüm 3.3'de anlatıldığı üzere hem düzenleyici, hem de görüntüleyici olarak kullanılmaktadır.

4 Kazanım

Cobalt kütüphanesi, test uygulamalarının kod tabanını küçülterek, zaman ve efor tasarrufu sağlamaktadır. Cobalt ile oluşturulan yazılım ürün hattı kazanımları Ergül'ün çalışmasında [12] incelenmiş ve nicel ölçümler alınmıştır. Bu bölümde, bu çalışmaya ait ölçüm ve değerlendirmeler, endüstriyel deneyimler ve kontrollü deneyler olmak üzere iki aşamada verilmektedir.

Endüstriyel Deneyim Sonuçları. Bu aşamada altı ayrı test uygulamasına ilişkin ölçümler alınmıştır. Her biri benzer ama farklı gereksinim seti ve geliştirme maliyetlerine sahip, test edilecek uygulamalarla TCP/IP protokolü ile haberleşmekte olan bu test uygulamaları, gerçek testlerde kullanılması amacıyla oluşturulmuştur. Tablo 1'de Cobalt'tan yararlanılmadan geliştirilmiş A, B, C



Şekil 2. Cobalt kullanılarak tanımlanmış bir veri tipinin grafiksel kullanıcı arayüzünde gösterimi ve düzenlenmesi

ve Cobalt ile geliştirilmiş D, E, F test uygulamalarına ait kod yeniden kullanım oranları verilmiştir. Yeniden kullanım oranı, yorum[23] ve anahtar kelimeler[24] içermeyen mantıksal kod satırları[25] içerisinde, tüm projelerde var olan kod satırlarının tüm kod satırlarına oranı olarak hesaplanmıştır.

Kontrollü Deney Sonuçları. İkinci aşamada, dört farklı test uygulaması kontrollü bir ortamda geliştirilmiş ve ölçümler alınmıştır. Tablo 2'de sunulan X ve Y test uygulamaları Cobalt kullanılmadan, X_{cobalt} ve Y_{cobalt} uygulamaları ise Cobalt'tan faydalanılarak geliştirilmiştir. X ve X_{cobalt} ile Y ve Y_{cobalt} uygulamaları kendi aralarında aynı gereksinim setlerine sahiptir.

İki Yönlü Geliştirme. Aselsan'da tasarlanan sistemler, birbiri ile haberleşebilen birden fazla yazılımdan oluşmakta ve bu yazılımların her birinin ayrı ayrı test edilmesi gerekmektedir. Bu yazılımlardan, aralarında veri alışverişi bulunan

[23] Comment
[24] Keywords. Ör: else, break, try, vs...
[25] Logical Lines of Code - LLOC

Tablo 1. Cobalt kütüphanesi kazanımlarını inceleyen endüstriyel deneyim sonuçları

	Kod yeniden kullanımı
A	%6
B	%14
C	%22
D	%85
E	%89
F	%94

Tablo 2. Cobalt kütüphanesi kazanımlarını inceleyen kontrollü deney sonuçları

	Kod yeniden kullanımı	Hata sayısı	Geliştime işçiliği[b]	İdame işçiliği[a] A	B	Kullanıcı memnuniyeti[c]
X	%55	12	11,73	33	47	51
Y	%79	4	1,5	29	75	
X_{cobalt}	%96	2	2,58	9	30	91
Y_{cobalt}	%97	2	0,38	26	45	

[a] Dakika cinsinden, geliştirici A ve B'ye ait idame maliyeti.
[b] Adam-Saat cinsinden.
[c] Simülatörlerin ortalama anket sonuçları.

her bir yazılım çifti için, aynı haberleşme protokolünün taraflarını gerçekleyen iki test uygulamasına ihtiyaç duyulmaktadır.

Yukarıda verilen sonuçlar, tekil test uygulamaları için geçerlidir. Cobalt'ın güçlü özelliklerinden biri de protokol tanımı üzerinden haberleşme mekaniğinin çift yönlü oluşturulabilmesi ve karşılıklı haberleşen iki test uygulamasının tek seferde ortaya çıkmasıdır. Bu sayede sistem geliştirme içerisindeki toplam efor yarıya inmektedir. Ayrıca bu özellik ile, geliştirilen test uygulamalarının kendi aralarında test edilebilirliği de kolayca sağlandığından, oluşturulan uygulamaların kalitesi artmaktadır.

5 Sonuç

Cobalt kütüphanesi, yazılım ve sistem testlerinde kullanılan test uygulamalarının geliştirme ve idame maliyetlerini düşürmek için tasarlanmış bir yapıtaşıdır. Bu kütüphanenin kullanımı öncesinde, son kullanıcıya ulaşmaması ve zaman endişeleri sebebiyle mimari tasarımlarına dikkat edilmeden hazırlanan test yazılımları, hem geliştirme, hem de idame esnasında fazla zaman ve uğraşa mal olmaktaydı. Cobalt kullanımı ile, bu uygulamaların oluşturulması ve güncellenen gerekler doğrultusunda değiştirilmesi kolaylaşmış ve hızlanmıştır.

Cobalt, Aselsan içerisinde 2010 yılından bu yana, 70'in üzerinde test uygulamasının geliştirilmesinde ve 30'un üzerinde yazılım/sistem testinde kullanılmıştır. Bölüm 2'de bahsedilen çözümlerle giderilemeyen ihtiyaçları karşılayarak şirket bünyesinde kabul görmüştür. Bununla birlikte, zamanla ortaya çıkan farklı gereklerin gerçeklenmesi ve hataların giderilmesi ile yeterli olgunluğa ulaşması sağlanmıştır. Özelleşmiş protokollerin tanımlanabilmesini sağlamanın yanı sıra, POJO/POJI eşleyici kullanılarak üçüncü taraflarca geliştirilmiş ara katman teknolojileri ile entegre olabilen, aktarılan verilerin gösteriminin ve düzenlenmesinin kolayca yapılabildiği, genişletilebilir bir kütüphane ortaya çıkmıştır.

Serileştirme ve ayrıştırma mekaniklerinin ve kullanıcı arayüzünün standardizasyonu, test uygulaması kalitesini arttırarak geliştirici kaynaklı hataların minimizasyonunu sağlamıştır. Ayrıca, birden fazla kişi tarafından gerçeklenen test uygulamalarında, geliştiriciler arasında ortak bir dil oluşturulmasına yardımcı olmuştur. Geliştirici faydalarının yanı sıra, az hatalı, homojen kullanıcı arayüzleri ve API katmanları ile kullanıcılarından da pozitif geribildirim alan Cobalt'la geliştirilmiş test uygulamaları, hızlı geliştirilebilmeleri sebebi ile test altındaki yazılımların hatalarının erken safhada bulunmasını sağlamıştır. Bölüm 4'te de nicel sonuçların gösterdiği üzere Cobalt kullanımı ile kaliteli, yüksek yeniden kullanımlı, geliştirme, idame ve test aşamaları kısalmış test uygulamaları elde edilebilmektedir.

6 Teşekkür

Yazarlar, kütüphanenin kullanımıyla elde edilen faydalar konusunda yaptığı araştırma için M. Erdem Ergül'e teşekkür eder.

Kaynaklar

1. Hunt, A., Thomas, D.: The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
2. Google, Protocol Buffers, `https://developers.google.com/protocol-buffers`
3. Apache Thrift, `http://thrift.apache.org`
4. CORBA, Object Management Group, `http://www.corba.org`
5. ASN.1, ITU-T Study Group 17, `http://www.itu.int/en/ITU-T/asn1`
6. Ragel, `http://www.complang.org/ragel`
7. Construct, `https://pypi.python.org/pypi/construct`
8. Protlib, `http://courtwright.org/protlib`
9. Google, Protocol Buffers, Encoding, `https://developers.google.com/protocol-buffers/docs/encoding`
10. Myers, B. A., Ronson, M. B.: Survey on User Interface Programming. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 195–202. ACM, New York (1992)
11. Google, Guice, `https://code.google.com/p/google-guice`
12. Ergül, M. E.: An Action Research of Achievements in a Software Product Line Implementation, MSc. Thesis, METU, Ankara (2014)