Gömülü Yazılım Testinde Farklı bir Yaklaşım: ScalaTest ile Test Otomasyon Aracı

Uğur YILMAZ[1] Ömer Faruk MORALIOĞLU[2]
[1,2,] Radar Elektronik Harp ve İstihbarat Sistemleri (REHİS) Grubu, ASELSAN A.Ş. Ankara
[1]uguryilmaz@aselsan.com.tr, [2]moralioglu@aselsan.com.tr

Özet. Gömülü yazılımlar, genellikle bir kullanıcı arayüzü sunmayan ve çevre yazılımlar ile yüksek hızlarda haberleşebilen yazılımlardır. Bununla birlikte, REHİS (Radar Elektronik Harp ve İstihbarat Sistemleri) bünyesinde geliştirilen gömülü yazılımlar, farklı algoritmalar için çeşitli ve yoğun miktarda girdi verilerine ihtiyaç duymaktadır. Bu özelliklere sahip yazılımların makul bir zaman içerisinde tüm arayüzlerinin ve algoritmalarının eksiksiz olarak test edilmesi ve bu testlerin sonuçlarıyla beraber raporlanması gerekmektedir. Bunun için test ve raporlama otomasyonu sık başvurulan ve verimli bir çözüm yoludur. Bu bildirinin konusunu teşkil eden çalışmada; ScalaTest'in sağladığı DSL aracılığı ile test adımlarını algılayabilen ve bu DSL üzerine farklı uygulama alanlarındaki gömülü yazılımlara uygulanabilen, geliştirilen benzetim yazılımlarını bir API ile kontrol edebilen ve test sonuçlarını aynı DSL aracılığı ile raporlayabilen bir test otomasyon aracı geliştirilmiştir. Ayrıca bu bildiride, geliştirilen aracın iki farklı alandaki gömülü yazılım testlerinde kullanımından elde edilen kazanımlardan bahsedilmektedir.

Anahtar Kelimeler Gömülü Yazılım Testi, ScalaTest Kütüphanesi, Test Otomasyon, DSL(Alana Özel Dil)

Abstract. Embedded software does not usually provide user interface and does interact with systems through high speed interfaces. In addition, the embedded software developed in REWIS (Radar Electronics Warfare and Intelligence Systems) requires various and highly-dense input data for diverse algorithms. It is obligatory that the tests of the software having above features should cover all algorithms and interfaces. Moreover the tests should be completed and reported in a reasonable amount of time. To achieve these goals, automation is a commonly applied and efficient method. This paper presents a ScalaTest based Test Automation Tool that can be adapted to different embedded software from various domains, can easily parse test steps through a human-readable DSL that is provided by ScalaTest, can control the simulated interfaces via an API and can report the test results using same DSL. As a verification, the proceeds of tests from two embedded software of different domains is presented as a case study.

Keywords: Embedded Software Test, ScalaTest Library, Test Automation, DSL

1 Giriş

Gömülü yazılımlar, gerçek zamanlı işletim sistemleri üzerinde çalışabilen, çevre yazılımlar ile arayüzlerinden yüksek yoğunlukta ve hızda verileri gerçek zamanlı olarak algoritmalarına besleyip sonuçlarını yine aynı arayüzlerden iletebilen türde yazılımlardır. Bu özelliklerinden dolayı Radar ve Elektronik Destek / Elektronik Taarruz (ED/ET) gibi çalışma alanlarında sıkça geliştirilmektedir. Gerçek zamanlı olmayan yazılımların yaratabileceği gecikmelerden uzak olması ve zaman kritik işlevleri daha kolay biçimde gerçekleştirmeleri bu çalışma alanlarında tercih sebebi olmaktadır.

Yazılımların bu karakteristiğinden ve endüstrinin hata toleransının düşük olmasından ötürü, yazılımlara ait testlerin de tüm haberleşme ve algoritma gereklerini kapsayacak şekilde geliştirilmiş olması gerekliliği doğmaktadır. Bununla birlikte yüksek çeşitlilikteki verilere ihtiyaç duyan bu gereklerin test edilmesi için manüel yöntemlerin kullanılması, proje takvimleri ve maliyet açısından büyük bir risk faktörü olarak açığa çıkmaktadır. Daha önceki çalışmalarda manüel testin otomatik teste oranla 2 kattan fazla zaman ve maliyet gerektirdiği görülmüştür [1]. Ayrıca manüel testlerde yazılım test tanımı ve yazılım test rapor dokümanlarının hazırlanması da önemli bir işçilik gerektirmektedir. Bu sebeple testlerin ve dokümantasyonun otomasyonu test mühendisliği açısından bir zorunluluk halini almaktadır.

Bununla birlikte, test otomasyonu da bir yatırım gerektirmekte ve bu yatırımın makul bir geri dönüşünün olması gerekmektedir [2]. Bu konuda bir araştırma yapıldığında karşımıza oldukça fazla sayıda araç veya kütüphane çıkmaktadır. Yapılan yatırımın karşılığını makul bir zamanda alabilmek ve test edecek olduğumuz yazılımın ve testlerimizin karakteristiğine en uygun araç ya da yöntemi belirlemek başlı başına bir çalışma konusu olmaktadır.

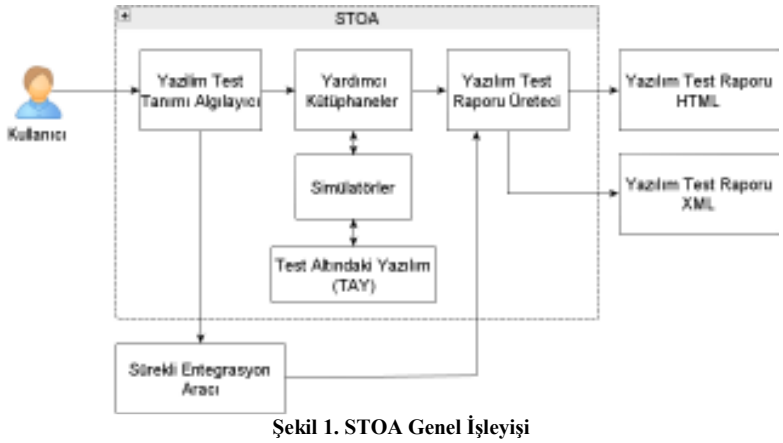
Yukarıdaki sorunları çözmek için ScalaTest'in sağladığı DSL özelleştirilerek bir arayüz geliştirilmiş ve bu arayüz sayesinde test adımlarını algılayabilen ve farklı uygulama alanlarına uygulanabilen bir test ve dokümantasyon otomasyon aracı, STOA (ScalaTest tabanlı Test Otomasyon Aracı), oluşturulmuştur. Bildirinin geri kalan kısmında öncelikli olarak test otomasyonu için kullanılacak kütüphane seçimi sürecinden, ScalaTest seçiminin altında yatan motivasyonlardan ve diğer bir test çerçevesi (framework) ile karşılaştırmalı analizlerden bahsedilecektir. Daha sonra bu kütüphane kullanılarak geliştirilen yardımcı test kütüphaneleri, nihai test otomasyon aracının geliştirme süreçleri ve işlevleri açıklanacaktır. Sonrasında Radar ve ED/ET çalışma alanlarında geliştirilmiş olan iki pilot yazılımın testlerinde bu kütüphane üzerinde geliştirilmiş olan otomasyon aracının kullanımından ve elde edilen kazanımlardan bahsedilecek ve sonuç olarak elde edilen deneyim ve gelecekte olası kullanım önerileri sunulacaktır.

2 Neden ScalaTest ?

Test aracının farklı alanlara uygulanmada kullanım esnekliği sunması önemlidir. Test aracının altyapısının buna elverişli olması gerekir. Bundan dolayı, yapılacak seçimde kütüphanenin geliştirilmesinin aktif olarak devam ediyor olması tercih sebebi olmaktadır. Aynı zamanda açık kaynak kodlu olması da güvenilirlik açısından avantaj sağlamaktadır. Böylelikle bu özelliklere sahip bir çerçeve üzerinde geliştirilecek araç hedef platformlara ve ihtiyaçlara göre geliştirme aşamasında şekillendirilebilecektir. Bu özelliklere sahip olan bir araç seçmek için şirket içinde kullanılan bir diğer çatı olan FitNesse [3] ile karşılaştırmalı analiz yapılmış ve ScalaTest seçilmiştir. ScalaTest'in hazır olarak sağladığı test işletim şablonlarında eksik olan ve geliştirilmesi gereken özellikler, şirket içinde kullanılan test süreciyle uyumlu olacak şekilde ScalaTest'in sağladığı DSL özelleştirilerek eklenmiştir. Böylece Yazılım Test Tanımları ve Raporları bu DSL kullanılarak üretilmiştir. Ayrıca kullanım kolaylığı ve özellik desteği gibi kriterler de ScalaTest'i öne çıkarmıştır.

ScalaTest, Scala ekosistemi içerisinde aktif olarak yeni yetenekler eklenen ve destek gören bir test aracıdır. 2013'te geliştirilmeye başlandığından beri ortalama ayda yaklaşık 80 kod değişikliği yapılması bunu doğrulamaktadır [4]. ScalaTest birçok yazılım aracıyla (JUnit, TestNG, Ant, Maven, Eclipse, Netbeans, IntelliJ vb.) birlikte çalışabilecek şekilde geliştirilmiştir. Asgari bir sözdizimi (syntax) öğrenimi ile test tanımlarının yazılmasına izin vermektedir. Bu sayede araçta eksik görülen, değiştirilmek istenen özellikler daha sonradan eklenebilmektedir. ScalaTest, test ihtiyaçlarını karşılamak için geliştirilmiş, isteğe göre değişebilen birçok küçük, odaklanmış araçlar sunan bir çalışma ortamıdır. [5] Bütün bu özellikler göz önünde bulundurularak ScalaTest değerlendirilmiş ve tüm bu yetenekler STOA'ya da kazandırılmıştır.

3 STOA

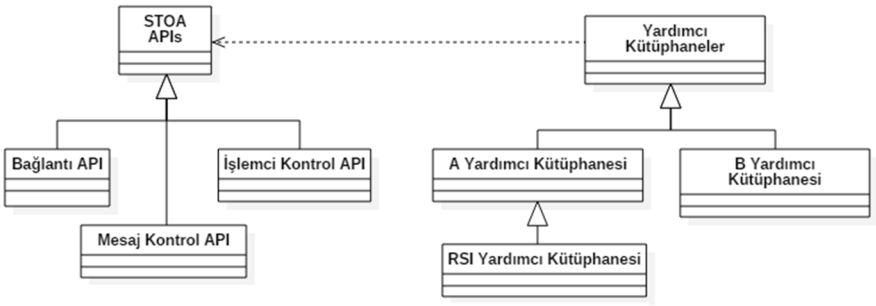


Şekil 1. STOA Genel İşleyişi

Şekil 1. STOA'nın genel işleyişini göstermektedir. STOA, yardımcı kütüphaneler ve test tanımı algılayıcı olmak üzere iki ana bileşenden oluşmaktadır. Kullanıcı test tanımlarını yazdıktan sonra STOA test tanımlarının içeriğini test tanımı algılayıcılarını kullanarak yardımcı kütüphanelere iletmektedir. Yardımcı kütüphaneler bir API aracılığıyla test altındaki yazılım (TAY) ile haberleşerek testleri çalıştırmaktadır. STOA'nın API altyapısı dışarıdan sağlanan simülatörler ile de kullanılabilir. Tekrar API aracılığıyla aldığı test sonuçlarını otomatik olarak raporlamaktadır. STOA'nın ana bileşenleri aşağıda açıklanmıştır.

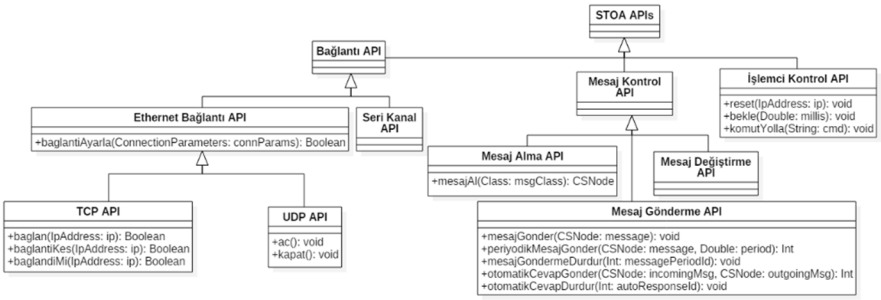
Yardımcı Kütüphaneler

Yardımcı kütüphaneler test tanım algılayıcıdan aldığı girdileri kullanarak API aracılığıyla testlerin koşturulmasını sağlar. Yardımcı kütüphaneler STOA'nın belirlediği bazı temel fonksiyonları gerçeklemek zorundadır. Bunlar Şekil 2.'de görülebileceği gibi Bağlantı, Mesaj Kontrol ve İşlemci Kontrol'den oluşmaktadır. Ek olarak, yardımcı kütüphaneler farklı uygulama alanlarına göre konfigüre edilebilir, amaca özel ek fonksiyonları gerçekleyebilir ve böylece sayıları birden fazla olabilir. Örneğin Şekil 2'de şirket içi geliştirilmiş olan benzetim altyapılarını kullanan yardımcı kütüphaneler görülmektedir. Uygulama esnasında amaca uygun olan yardımcı kütüphane seçilerek kullanılabilir. Böylece genişletilebilir modüler bir mimari yapı sunulmaktadır.



Şekil 2. STOA UML Sınıf Diyagramı

Şekil 3'de STOA ile varsayılan olarak sağlanan yardımcı kütüphanenin gerçeklemesi gösterilmiştir. Görüldüğü gibi birden fazla haberleşme protokolü, mesaj alma/gönderme ve işlemci kontrol yöntemleri desteklenmektedir. Böylece farklı uygulama alanlarının ihtiyaçlarına göre istenen özellikler kullanılabilir.



Şekil 3. Örnek Yardımcı Kütüphane Altyapı Diyagramı

Test Tanım Algılayıcı

Test Tanım Algılayıcı, kullanıcının oluşturduğu test tanımlarını girdi olarak alıp ScalaTest yardımıyla Yardımcı Kütüphanelere iletir. Kullanıcının yazılım test tanımlarını oluştururken şirket içi belirlenen yazılım test tanım doküman şablonuna uyulması önerilir. Bu şablon için gereken alanları Test Tanım Algılayıcı, ScalaTest'in sağladığı DSL'i özelleştirerek oluşturur. Yardımcı Kütüphanelerde olduğu gibi farklı uygulama alanlarında kullanılmak üzere farklı yazılım test tanım şablonları yaratılabilir.

```
class IsımaBaslatTesti extends YTET with RSITestHelper {
  yorum("Bu test Radar Sinyal İşleme testinde bulunan Isıma Baslat
Mesaji testi islemlerini gerceklestirmektedir.")
  testAmaci("Yazılımın açılıştan sonra İsima islemlerinin
dogrulanmasi.")
  testtenOnce() {
    bekle(5 saniye)
    controller baglantiKur }
  testtenSonra()  {
    yorum("Gerekli işlemler için 5 saniye bekle”)
    bekle(5 saniye)
    controller baglantiKes }
  testYonergesi() {
    adim("Acilis islemleri tamamlandiktan sonra Simulatorden Hazır
Mesajı gonderilir.") {
      val hazir = new HazirMsg
      hazir durum ISIMA_BASLAT
      controller mesajGonder hazir }
    beklenen("Yazılımın Simülatöre gelen Isıma Baslat mesajini dogru
olarak olarak gonderdigi gozlenmelidir.") {
```