Gerçek Zamanlı Sistemlerde Otomatik Test Yaklaşımı

Automated Test Approach in Real Time Systems

Ömer Karaduman
Yazılım Test ve Yönetim Birimi
ASELSAN MGEO, Ankara
okduman@mgeo.aselsan.com.tr

Celal Ademoğlu
Yazılım Test ve Yönetim Birimi
ASELSAN MGEO, Ankara
ademoglu@mgeo.aselsan.com.tr

Emrah Mert
Yazılım Test ve Yönetim Birimi
ASELSAN MGEO, Ankara
emmert@aselsan.com.tr

Özet

Yazılım projelerinde sistem seviyesi doğrulama testleri kritik bir öneme sahip olmakla birlikte genelde bu testler için projelerdeki zaman darlığı ve planlamadaki yetersizlikler sebebi ile geliştirme ve entegrasyon için ayrılan zamandan çok daha az bir zaman kalmaktadır[1]. Bu da ortaya doğru ve hatasız bir yazılım ürünü çıkarma yolunda zamanı etkin bir şekilde kullanmanın önemini bir kat daha arttırmaktadır[2]. Bu sebeple ki yazılım testlerinin otomatik yapılmasındaki ilk amaç testlerin elle yapıldığı takdirde sarf edilen zaman ve işgücünü asgariye indirmek olmuştur. İkinci bir amaç ise ortaya çıkacak hataların kısa bir sürede tespit edilerek düzeltici işlemlerin yapılmasının ardından tekrar teste tabi tutulması ve yine hızlı bir şekilde düzeltildiğinin görülmesidir. Bir diğer amaç ise DO-178B gereği yapılması gereken ve gereksinim tabanlı testlerin yazılımın yapısının ne kadarını çalıştırdığını anlamaya yarayan yapısal kapsama analizleri için otomatik koşturulan testlerin büyük fayda sağlamasıdır. Bütün bu amaçlar göz önünde bulundurularak ASELSAN MGEO bünyesinde yazılım test otomasyonu altyapısı geliştirilmiş ve ilk uygulaması T-38 MKB (Merkezi Kontrol Bilgisayarı) Harekat Uçuş Yazılımı testlerinde kullanılmıştır. Bu makalede, geliştirilen test otomasyon altyapısının temel prensiplerinden ve getirdiği kolaylıklardan bahsedilecektir.

Abstract

Although system level verification tests of software projects have a critical importance, the time left for them are usually much less than the time spent in development and entegration because of lack of time inadequacy of planning in projects. This doubles the importance of using time more efficiently in order to have a correct and error-free software product. For this reason, the first objective for having the software tests automated had been reducing the time and labor that had been spent running the tests manually. Second objective is to detect the errors in short notice and running regression tests quickly in order to prove that the errors are corrected as expected. Another motivation is the advantage of automated tests in getting the code coverage of the requirement based tests which is mandatory for complying DO178B standards. Taking all these into consideration, a test automation framework had been developed in ASELSAN MGEO and it was first applied in system tests of T-38 CCC Operational Flight Software. In this article, basic principles and the advantages of the developed test automation framework is explained.

1. Giriş

Yazılım ürünlerinin otomatik olarak test edilmesi yeni bir kavram olmamakla birlikte gerçek zamanlı gömülü yazılımların sistem seviyesindeki testlerinin otomatikleştirilmesi üzerine yaygın çalışmalar bulunmamaktadır. 1990'lı yılların başlarından beri var olan otomasyon çalışmaları[3], temel olarak masaüstü programları ve internet sayfalarının testleri konularına odaklanmış, gömülü sistemlerin testleri ile ilgili, genel geçer sistemler ortaya konulamamıştır. Bu durum, gömülü sistem testlerini otomatik hale getirmek isteyen kurumların, kendi test altyapılarını oluşturma gerekliliğini ortaya çıkarmıştır.

Masaüstü ve internet uygulamaları için tasarlanan test otomasyon yaklaşımlarının gömülü yazılımlarda neden kullanılamadığını açıklamak için, mevcut otomasyon sistemlerini şu şekilde gruplayabiliriz [4]:

- Kaydet/Oynat tipindeki testler
- Yeniden kullanılabilir test betikleri
- Veri güdümlü test betikleri
- Anahtar kelime güdümlü betikler
- Betiksiz otomasyon sistemleri

Kaydet/Oynat tipindeki test otomasyon sistemlerinin gömülü sistemlerin testlerinde kullanılamamasına sebep olan pek çok etken vardır. En temel farklılık, gömülü yazılımlarda çoğunlukla, yazılım tarafından erişilebilecek bir görsel arayüzün olmamasıdır. Varolan görsel arayüzlerin sistemle haberleşmeleri de karmaşık mesajlaşmalar içermekte ve çoğu zaman bütün veri girişleri için yeterli altyapıyı sağlayamamaktadır.

Betik esaslı test otomasyon sistemleri ise testleri tanımlayacak olan kişilerin, kullanılan betiğe belli seviyede hâkim olmalarını gerektirmektedir. Anahtar

kelime güdümlü betikler bu aşamayı kolaylaştırsa da gömülü sistemlerin veri alış verişinde bulunduğu çok sayıda farklı uç birim bulunabileceği için anahtar kelime sayısı artacak, bu da testler hazırlanırken karışıklıklara yol açabilecektir.

Belirtildiği üzere gömülü sistemler, günlük uygulamaların aksine pek çok kaynaktan beslenmekte ve farklı uç birimlere veri sağlamaktadır. Gömülü bir sistemin otomatik olarak test edilebilmesi için ya bu uç birimlerin yerine gerçek cihazlar kullanılmalı ya da benzetimler tanımlanmalıdır. Uç birimlerin yerine gerçek cihazların kullanılması durumunda sisteme gönderilmek istenen bütün veriler sağlanamayabileceği ve sistemden gelen bütün veriler gözlemlenemeyebileceği için benzetimlerin tanımlanması en doğru çözüm olmaktadır. Bu durumda da, gömülü sistemin kullandığı bütün haberleşme protokolleri ve mesajlaşma tanımlarına uygun benzetimlerin tanımlanması gerekmektedir.

Gerçek zamanlı sistemlerin zaman kısıtlamaları, günlük uygulamalardaki zaman kısıtlamalarına göre çok daha sıkıdır. Kullanılacak olan otomasyon sisteminin bu kısıtlamalara uygun hareket edebiliyor olması en önemli şartlardan birisidir.

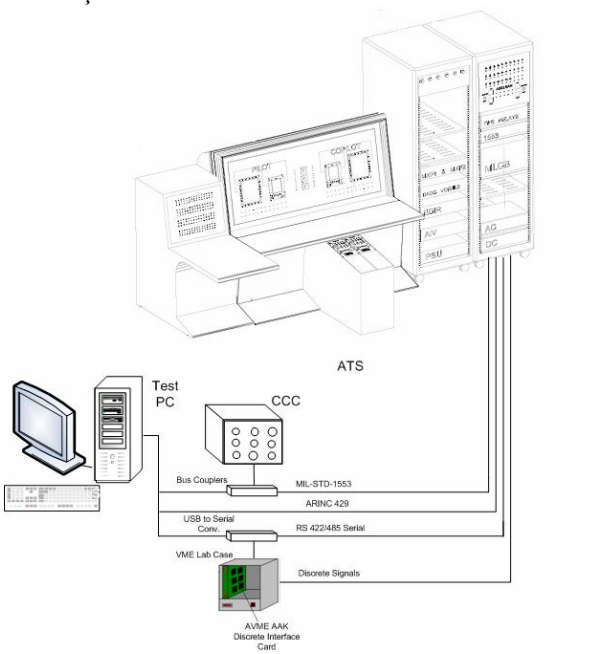
Bunların yanısıra, askeri standartlarda geliştirilen gömülü sistemlerin testlerinde, koda bir müdahalede bulunulması veya test programının hedef sistem üzerinde koşturulması, test edilen sistem ve gerçek sistem arasında farklılıklara sebep olacağı için böyle bir yöntem izlenmesi de mümkün değildir.

Bütün bunlar göz önüne alındığında, mevcut otomasyon yaklaşımlarından gömülü yazılımlara en uygun olanı "betiksiz otomasyon sistemleri" gibi görülse de piyasada hazır olarak bulunabilecek betiksiz otomasyon sistemlerinin hedef uygulama aralığı çok dardır. Neticede, günlük uygulamalarda kullanılan yöntemlerin, gerçek zamanlı sistemlerin testlerinde doğrudan kullanılmasının pek mümkün olmadığı görülebilmektedir.

ASELSAN bünyesinde geliştirilmekte olan sistemler genel olarak gerçek zamanlı gömülü sistemler olduğu için bu sistemlerin testlerini otomatik hale getirebilecek bir altyapı oluşturulmasının gerekliliği anlaşılmış ve konuyla ilgili çalışmalar neticesinde başarılı sonuçlar elde edilmiştir. Makalenin devamında, ASELSAN MGEO Yazılım Doğrulama Ekibi olarak gerçek zamanlı sistemlerin testlerinin otomatik hale getirilmesi için getirdiğimiz yaklaşımdan bahsedilecek, karşılaşılan sorunlar ve bu sorunların çözümleri incelenecek, elde edilen kazanımlar ve yaklaşımımızın nasıl geliştirilebileceği ile ilgili bilgiler verilecektir.

2. Genel Yaklaşım

Bu bölümde, test otomasyon altyapısında göz önünde bulundurulan temel özelliklerden bahsedilecektir. İlk bölümde kullanılan programlama dilleri ve yazılım çatıları açıklanacak, sonrasında kullanıcı arayüzünün tasarımındaki esaslardan bahsedilecek, devamında haberleşme arayüzlerinin nasıl tanımlandığı açıklanacak ve temel çalışma şeklinin üzerinden geçilecektir. Testlerin otomatik olarak koşturulması demek test ortamının da buna uygun olarak hazırlanmasını gerektirmektedir. Şekil 1'de bu bildirinin örnek uygulaması olan MKB/HUY yazılım testlerinin koşturulduğu ortam verilmektedir. Burada otomatik test uygulaması Test PC üzerinde koşarken test edilen hedef (CCC/OFP-MKB/HUY) ile çeşitli arayüzler üzerinden gerçek zamanlı olarak haberleşmektedir.



Şekil 1 MKB/HUY Yazılım Test Ortamı

2.1. Kullanılan Programlama Dilleri ve Yazılım Çatısı

Test otomasyon sistemi temel olarak .NET 2.0 çatısı üzerinde, Microsoft Visual C# kullanılarak geliştirilmiştir. Test sonuçlarının raporlanması için "Microsoft.VisualStudio.TestTools.UnitTesting" isim uzayı altındaki "Assert" sınıfları kullanılmış, bu sayede test sonuçlarının Microsoft Visual Studio dahilindeki test koşturma arayüzünde de incelenebilmeleri sağlanmıştır.

Tanımlanan test adımları, CodeDOM isim uzayı aracılığı ile Managed C++ koduna dönüştürülmüş ve testler, bu kodun derlenmesiyle üretilen uygulamanın çalıştırılması ile koşturulmuştur.

2.2. Kullanıcı Arayüzü

2.2.1. Farklı bir arayüz gereksinimi

Masaüstü ve internet uygulamalarının test otomasyonunu sağlayan sistemlerde testler, uygulamanın kendi arayüzü aracılığıyla tanımlanabilmektedir. Testi girecek olan kişiler, altyapıda mevcut olan mesajlaşmalardan haberdar olmak zorunda değildir, test edilecek olan uygulamaya hakim olmak yeterlidir. En basit ve yaygın olarak kullanılan test otomasyon sistemlerinde arayüz üzerinden yapılan veri girişi ve tuş basımı gibi işlemler kaydedilerek test durumlarına dönüştürülür ve bu test durumları, kullanıcı etkileşimine gerek duymaksızın yeniden koşturulabilir

Gerçek zamanlı sistemlerde ise veri giriş çıkışları çoğu zaman kullanıcıların doğrudan kontrolünde değildir. Kullanıcının giriş yapabildiği arayüzler test edilen sistemle karmaşık mesajlar aracılığı ile haberleşirken görsel olmayan arayüzlerde zaten kullanıcının bir müdahalesi mevcut değildir. Veri giriş çıkışlarının kontrol edilmesi ancak en alt seviyede, kullanıcının test edilen sisteme giden ve bu sistemden gelen mesajları kontrol etmesi ile mümkün olmaktadır. Bu durumlar göz önüne alındığında, testlerin girilmesi için özel arayüzlerin hazırlanmasının gerekliliği ortaya çıkmaktadır.

Hazırlanacak arayüzler benzetimler (simülatör/emülatörler) şeklinde olabilir. Bu tür benzetimlerin hazırlanması halinde kullanıcı, testleri simülatörler aracılığı ile yapabilir ve testler, masaüstü uygulamalarındaki gibi kaydedilebilir. Bu yöntem ilk bakışta makul gibi dursa da gerçek zamanlı sistemlerin testleri için uygun değildir. Öncelikle, bütün haberleşme arayüzleri için ayrı benzetim programlarının hazırlanması gerekecektir. Bu benzetimlerin kullanılmadan önce ayrıca test edilmeleri ve beklenildiği şekilde çalıştıklarının doğrulanmaları gerekecektir. Bunun yanısıra, hedef sistem gerçek zamanlı bir uygulama olduğu için benzetimlerin çalışma zamanlamalarının birbirlerine uydurulmaları gerekecektir.

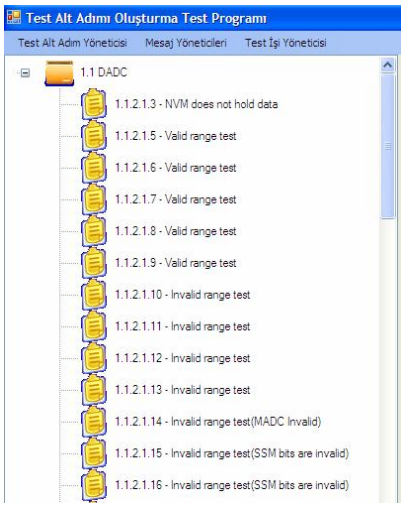
Diğer bir yaklaşım ise bütün benzetimlerin tek bir uygulama altında toplanması ve kullanıcıya mesajlaşmaları tek bir arayüzden girebilme imkanı sağlanmasıdır. Bu sayede benzetimlerin birbirleriyle uyumlu çalışmasının sağlanması daha kolay olacağı gibi uygulamanın doğru çalıştığının doğrulanması da daha kolay olacaktır.

ASELSAN MGEO bünyesinde geliştirilen test otomasyon altyapısında ikinci yöntem olan benzetimlerin tek bir uygulama altında toplanması yöntemi izlenmiştir. Bir sonraki bölümde bu arayüzden örnekler verilerek genel arayüz mantığı açıklanmaya çalışılacaktır. Daha sonraki bölümde ise bu arayüz aracılığı ile hazırlanan mesajların işlenmesini sağlayan haberleşme elemanlarından bahsedilecektir.

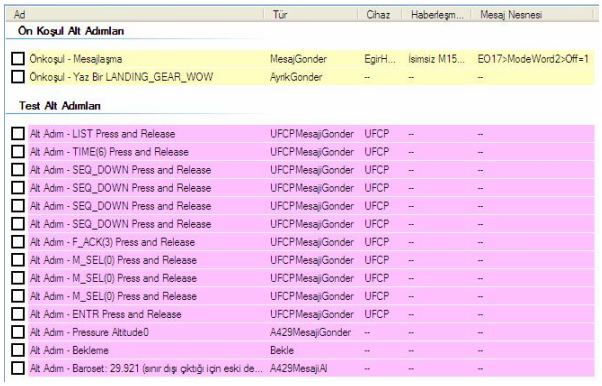
2.2.2. Test tanımlama arayüzü

Hazırlanan kullanıcı arayüzünün, test tanımlayacak olan kullanıcıların, alt seviyedeki mesajlaşma yapıları hakkında bilgileri olmadan da test tanımlayabilmelerini sağlaması gerekmektedir. Bunu sağlayabilmek için benzetimleri hazırlanmış olan cihazların arayüz kontrol dokümanlarında tanımlanmış olan mesaj yapıları, birebir olarak test arayüzüne aktarılmalıdır. Ayrıca, bu mesaj yapılarına ulaşımın kolaylaştırılması için kullanıcıya çeşitli filtreleme seçenekleri sunulmalıdır. Şekil 2 (a, b, c)'de hazırlanmış olan arayüzlerden örnek bir ekran, üç kısım halinde görülebilir.

Hazırlanan arayüzde, bu filtreleme bir kaç seviyede yapılmıştır. Öncelikle, ortak bir haberleşme protokolu kullanan mesajlar birbirlerinden ayrılmıştır. Bu gruplama neticesinde MIL-STD-1553 ve ARINC-429 gibi ana arayüzler ayrılırken kendine has mesajlaşma sistemleri içeren ayrık sinyaller, çok işlevli gösterge (Multi Functional Display – MFD), üst ön kontrol paneli (Up Front Control Panel – UFCP) gibi sistemler ayrıca gruplanmıştır.



(a) Test adımları listesi genel görünümü

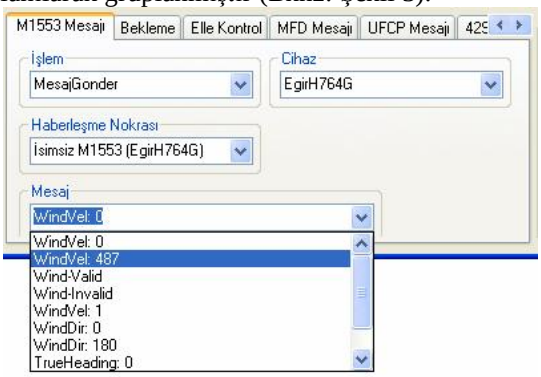


(b) Test altadımları listesi genel görünümü

(c) Altadım ekleme mönüsü genel görünümü

Şekil 2 Test adımı tanımlama arayüzü genel görünümü

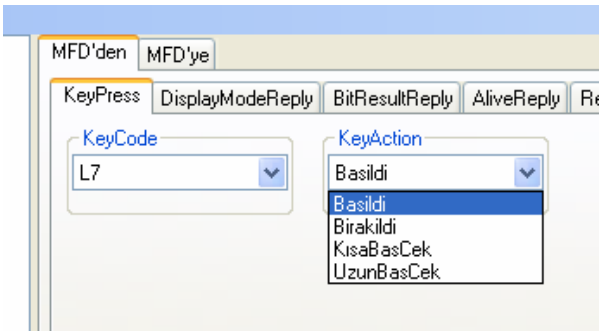
Test edilen sistemle haberleşen ve MIL-STD-1553 ile ARINC-429 protokollerini kullanan birden fazla cihaz olduğu için bu gruplar altında öncelikle cihaz bazlı filtreleme imkanı sağlanmıştır. MIL-STD-1553 arayüzü kullanan cihazların altlarındaki filtreleme alt adresler ve kelime numaraları ile sağlanırken ARINC-429 cihazlarına ait mesajlar, mesaj etiketleri kullanılarak gruplanmıştır (Bknz: Şekil 3).



Şekil 3 MIL-STD-1553 mesajlarının gruplanması

Çok işlevli gösterge ve üst ön kontrol paneli cihazlarına ait mesaj yapılarının tanımlanabilmesi için bu cihazların sağladığı, arayüzde görünen ve görünmeyen bütün işlevler kullanıcı arayüzüne aktarılmıştır (bknz: Şekil 4). Bu sayede, test girişi yapan kullanıcıların, alt seviyedeki mesajlaşma detaylarından sıyrılırken normalde gerçek cihazlar aracılığıyla gerçekleştiremeyecekleri hatalı mesaj oluşturma, cihaz içi test sonucu gönderme gibi işlevleri de basit bir arayüz kullanarak tanımlayabilmeleri sağlanmıştır.

Test adımları tanımlanırken, bütün mesajların her bir alt alanı için "Gözardı Et" seçeneği eklenmiştir. Bu seçenek sayesinde kullanıcı, her bir test adımında, ilgili cihazın gönderilecek veya doğrulanacak mesajındaki bütün alt alanları doldurmak zorunda kalmamaktadır.



Şekil 4 Çok işlevli göstergelerin mesaj tanımlama arayüzünden bir kesit

Gözardı edilmek üzere işaretlenmiş olan alanlar doğrulama adımlarında dikkate alınmazken gönderme adımlarında, o altalandaki mevcut değerin korunmasını sağlamaktadır.

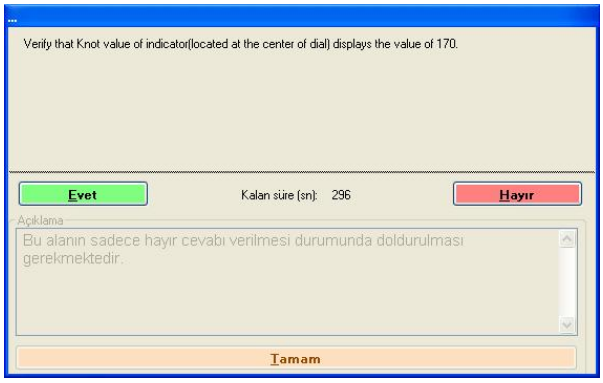
Cihazların benzetimlerinin aktarılmalarının yanısıra, genel amaçlarla kullanılabilecek test adımlarının tanımlanabilmeleri için "Bekleme" ve "Elle Kontrol" şeklinde isimlendirilmiş iki ayrı test adımı türü tanımlanmıştır. "Bekleme" adımları sayesinde, test adımları arasına milisaniye hassasiyetinde beklemeler tanımlanabilmesi, bu sayede gerçekleşmesi beklenen işlemler için belli bir süre geçmesi gerekmesi durumunda gerekli beklemenin oluşturulabilmesi sağlanmıştır. "Elle Kontrol" adımları ise doğrulama yazılımı tarafından otomatik olarak gerçekleştirilemeyen veya testi koşturan kişi tarafından elle yapılmasının daha kolay olduğu test adımlarının yapılabilmesi için kullanıcıya yönergeler girilebilmesi sağlanmıştır. Test koşturulurken bir "Elle Kontrol" adımına gelindiğinde ekranda, kullanıcı tarafından test hazırlanırken girilmiş olan yönerge, yine kullanıcı tarafından tanımlanmış süre boyunca gösterilmektedir. Yönergede tarif edilen işlem veya doğrulama aktivitesi başarıyla tamamlandığında kullanıcı test adımını "Geçti" olarak işaretleyebilmekte, bir hata olması durumunda test adımı "Kaldı" olarak işaretlenip gerekli açıklama yine bu arayüzden girilebilmektedir. Şekil 5'te, örnek bir elle kontrol penceresi görülebilir.

Test adımı tanımlamanın yanı sıra, testlerin gereksinimlerle olan bağlantılarını kurabilmek için kullanıcılara, her bir test adımının Telelogic Doors ID'sini girebilecekleri arayüzler sağlanmıştır. Bu

sayede koşturulan test adımlarının Doors ortamındaki eşsiz ID'leri kullanılarak sonuç raporlama işlemleri yapılabilmektedir.

2.3. Haberleşme arayüzleri

Test adımlarında kullanılacak bütün benzetimler için haberleşme arayüzleri tanımlanmıştır. Tanımlanan arayüzler, haberleşmelerin sağlıklı yürütülebilmesi için benzetimler içerebildiği gibi yalnızca mesaj alma-gönderme işlevlerini tanımlayan fonksiyonlar da olabilmektedir.



Şekil 5 Elle kontrol penceresi

Benzetim gerektirmeyen arayüzler arasında MIL-STD-1553, ARINC429 gibi genel protokollerin yanı sıra Bekleme ve Elle Kontrol tipindeki test adımları sayılabilir.

- **MIL-STD-1553:** Bu mesajlar, hedef bilgisayardaki veri yolu yöneticisi tarafından sürülmektedir. Test arayüzü tarafından veriyoluna yazılan bir değer, fazladan bir işlem yapılmaksızın gönderilmeye devam edilmektedir. Bu yüzden, özel bir el sıkışma gerektirmeyen durumlarda MIL-STD-1553 mesajlaşmaları için özel bir benzetim yapılmasına gerek kalmamaktadır[4].
- **ARINC-429:** ARINC-429 mesajlarının tazeliklerinin korunabilmesi için bu mesajların önceden belirlenmiş periyotlarda veriyoluna yazılması gerekmektedir. Bunun haricinde bir işlem yapılmasına gerek kalmadığı için özel bir benzetim hazırlamak şart değildir
- **Bekleme:** Bekleme mesajları, arkaplandaki periyodik haberleşmeleri ve benzetimleri kesmeden test adımlarını duraklatma dışında bir işlev gerçekleşmediğinden ayrı bir benzetime ihtiyaç duyulmamaktadır.
- **Elle Kontrol:** Elle kontrol mesajları da bekleme mesajlarına benzemekle beraber farklı olarak işlem yönergesi ve test sonucu parametreleri içermektedir. Bu işlev de ayrı bir benzetim olmaksızın çalışabilmektedir.

Benzetim gerektiren haberleşme protokolleri için gerekli olan sistem, daha önce de belirtildiği üzere test aracının içine dâhil edilmiştir. Bu benzetimler, hedef sistemin sorgu-cevap şeklinde tanımlanmış, sürekli olarak devam eden, gecikmelerin veya hatalı cevap mesajlarının sistemin beklenmeyen bir hale gelmesine sebep olabileceği durumları kontrol altında tutabilmek amacıyla oluşturulmuştur.

Çok işlevli gösterge ve üst ön kontrol paneli mesajlaşma arayüzleri, periyodik bir sorgu cevap yapısıyla çalışmaktadır. Buna göre, hedef bilgisayar gerekli komut mesajlarını üretmekte ve karşılığında, biçimi önceden belirlenmiş mesajlar beklemektedir. Bu uç birimler için tasarlanan benzetimler, düzenli olarak hedef bilgisayardan gelen sorguları dinlemekte, kullanıcının test adımlarında kullandığı değerleri kullanarak cevap mesajları oluşturmakta ve bu mesajları göndermektedirler.

Daha önce, MIL-STD-1553 mesajlarının genel olarak benzetime ihtiyaç duymadığından bahsedilmişti. Test otomasyon altyapısının ilk kez uygulandığı proje olan T-38 MKB Harekat Uçuş Yazılımı Geliştirme projesinde, MIL-STD-1553 veriyolunu kullanan fakat yoğun bir benzetim gerektiren bir veri yükleme/kaydetme uç birimi de mevcut olduğu için, sadece bu uç birime ait mesajlara özel bir benzetim tanımlanmıştır. Bu benzetim sayesinde dosya kaydetme ve yükleme işlemleri sekteye uğramadan devam ettirilebilirken kullanıcılar, giden gelen mesajlara hata zerk ederek sistemin, hata durumlarına tepkilerini de ölçebilmişlerdir.

Farklı bir benzetim de ayrık mesajların yazılıp okunabilmesi için tanımlanmıştır. Ayrık sinyalleri yazılım aracılığı ile doğrudan kontrol etmek mümkün olmadığı için üzerinde röleler bulunan bir ayrık arayüz kartı kullanılmış ve bu kartla haberleşebilmek için gerekli el sıkışma mesajlarının otomatik olarak oluşturulup işlenebilmesi sağlanmıştır.

Görülebildiği üzere, tek uygulama üzerinde birden fazla benzetim tanımlanmış, hatta aynı protokol üzerinden hep benzetimli hem de benzetimsiz haberleşme arayüzlerinin aynı anda çalıştırılabilmesi sağlanmıştır.

2.4. Çalışma prensipleri

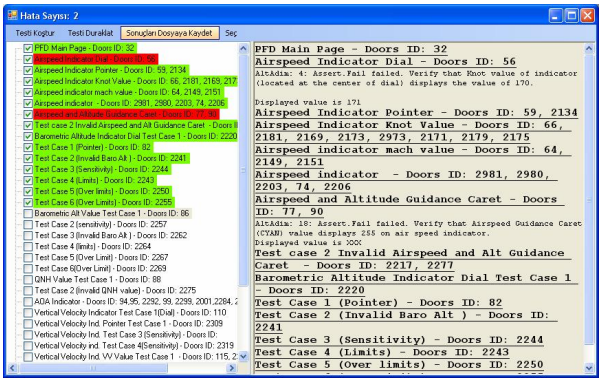
Test otomasyon altyapısı temel olarak iki ana arayüzden oluşmaktadır. Bunlardan ilki testlerin tanımlamasını sağlarken ikincisi testlerin koşturulmasını sağlamaktadır.

Bu şekilde iki ayrı arayüzün hazırlanmasının temel sebebi, C# ve C++'ın farklı avantajlarından yararlanabilmek olmuştur.

Haberleşmelerde kullanılacak olan mesaj yapıları, C yapıları (struct) olarak kolaylıkla tanımlanabilmektedir. Aynı yapılar C#'ta tanımlanmak istendiğinde baytların hafızadaki dizilimini düzenlemek için fazladan işlemler yapmak gerekmekte, bu da performansı olumsuz yönde etkilemektedir. Diğer bir yandan C#, daha az kod parçası ile daha çok iş yapmaya olanak sağlamakta, bu da yazılımın daha hızlı ilerlemesine yardımcı olmaktadır.

Uç birimlerin mesaj yapıları, Rhapsody üzerinde C yapıları olarak tanımlanmıştır. Test uygulaması, Rhapsody üzerinden bu yapıları çekerek öncelikle C# tarafından algılanabilecek sınıflar oluşturmakta, bu sınıfları uygulamaya dinamik olarak aktarmakta ve kullanıcıların bu sınıfları kullanarak testler tanımlayabilmelerini sağlamaktadır. Testler koşturulacağı zaman, Microsoft Visual Studio'nun sunduğu CodeDOM altyapısı kullanılarak Managed C++ kodu oluşturulmaktadır. Bu kod temelde C++ kodu olduğu için Rhapsody tarafından üretilen başlık (header) dosyalarını kullanabilmekte, Managed C++ kodu olduğu için de C#'la yazılmış sınıflara erişebilmektedir. Bu sayede hem performanstan ödün verilmemiş, hem de arayüzler çok hızlı bir şekilde geliştirilebilmiştir.

Oluşturulan C++ kodu, yine test arayüzü aracılığı ile derlenip çalıştırılabilmektedir. Derleme sonucunda, koşturulabilecek testlerin bir ağaçta sıralandığı ve test sonuçlarının izlenebildiği ikinci arayüz hazır hale gelmektedir. Bu arayüzde seçilebilecek test adımları, C++'ın kodundaki fonksiyonlara denk gelirken fonksiyonlardaki her bir kod deyimi, kullanıcı tarafından tanımlanan test adımlarını temsil etmektedir.



Şekil 6 Test koşturma arayüzü

Şekil 6'da gösterilen test koşturma arayüzü çalıştırıldığı zaman öncelikle, testlerde kullanılan benzetimler çalıştırılmaktadır. Test adımlarındaki mesajlaşmalar doğrultusunda mesaj alış verişleri düzenlenmekte, doğrulama adımlarının sonuçlarına göre test adımının geçti/kaldı bilgisi kullanıcıya iletilmektedir.

Test sonuçları, "Zengin Metin Biçimi"nde (Rich Text Format, RTF) kaydedilip incelenmek üzere saklanabilmektedir.

3. Karşılaşılan Zorluklar

Karşılaşılan en temel zorluk, hedef sistem gerçek zamanlı bir işletim sistemi üzerinde koşarken test sisteminin Microsoft Windows üzerinde koşuyor olması olmuştur. Test bilgisayarının gerçek zamanlı çalışmıyor olması ve aynı anda pek çok servis ve uygulamanın koşturuluyor olması test uygulamasının performansını olumsuz yönde etkilemiştir. Bu olumsuzluğu gidermek için test uygulaması içerisinde oluşturulan her bir ilmeğe yüksek öncelik verilmiş, bu sayede işlemciyi diğer uygulamalarla paylaşıyor olmanın getirdiği olumsuzluklar asgariye indirilmiştir.

Kod içerisinde ise haberleşme elemanları ve benzetimler tek tek incelenmiş, algoritmalar yeniden düzenlenmiş ve kodun olabildiğince hızlı şekilde çalışması sağlanmıştır.

Benzetimlerden seri kanal kullananlar için .NET çatısı tarafından sağlanan SerialPort sınıfı yerine daha alt seviye yöntemler kullanılmış, bu sayede seri haberleşmelerde önemli bir performans artışı sağlanmıştır.

Bir diğer sorun, çok işlevli göstergeler üzerindeki verilerin doğrulanması olmuştur. Bu doğrulamalar "Elle Kontrol" olarak tanımlanmış ve testi koşturan kişi tarafından, çok işlevli gösterge ekranlarına bakmak suretiyle doğrulanmıştır. Bu durum, testlerin koşturulma hızında bir darboğaz oluşturduysa da bu tipteki görsel doğrulamaların otomatik yapılması için gerekli olan altyapıyı hazırlamak için geçecek süre, projenin test takviminde sıkıntı yaratacağı için bu

doğrulamaların elle yapılmaya devam edilmesi uygun görülmüştür [6]. Test aracının daha sonraki sürümlerinde bu tipteki doğrulamaların da otomatik yapılması planlanmaktadır.

Ayrıca, test altyapısı ve bu altyapının kullanılması planlanan hedef sistemin testlerinin yazılması işlerinin paralel gitmesi gerektiği için testler öncelikle elle, düz metin olarak tanımlanmış, daha sonra bu testler hazırlanan arayüze girilmiştir. Bu durum, testler tanımlanırken fazladan iş gücü harcanmasına sebep olmuşsa da test aracında yapılan geliştirmeler sayesinde bu sorun ortadan kaldırılmış, test altyapısı kullanılarak tanımlanan testlerden otomatik olarak test dokümanlarının oluşturulması sağlanmıştır.

4. Kazanımlar

Daha önce de belirtildiği üzere test otomasyon altyapısının kurulmasındaki en büyük motivasyon, testlerin koşturulmasında harcanan zaman ve iş gücünün azaltılmasıdır. T-38 MKB Harekat Uçuş Yazılımı için gereksinimler esas alınarak hazırlanan yaklaşık 12000 adet doğrulama adımı, 5 gün gibi kısa bir sürede koşturulabilmiştir. Bu testler koşturulurken, çok işlevli göstergeler üzerindeki görsel öğelerin doğrulanması dışında bir iş gücü harcanılmamıştır.

Testler koşturulurkenki insan müdahalesi asgari düzeyde tutulduğu için testlerin tekrar edilebilirliği azami seviyede olmuştur. Elle koşturulan testlerde, özellikle zamanlama farklılıklarından kaynaklanan küçük hatalar test sonucunun farklı çıkmasına sebep olurken test otomasyon altyapısının kullanımı sayesinde bu tür sorunlarla karşılaşılmamıştır.

Hazırlanan bu altyapının gerçek bir projede uygulanması neticesinde geliştirilebilecek noktaları tespit edilmiş ve bu altyapının daha genel geçer hale getirilmesi için çalışmalar yapılmıştır.

Test aracına girilen testlerin karakteristikleri incelenerek, koşturulacak testlerin otomatik oluşturulması ile ilgili çalışmalar yapılmıştır. Çeşitli cihazlara ait arayüz kontrol dokümanları otomatik olarak içe aktarılıp kullanıcının tanımladığı test taslaklarına uygun testler otomatik olarak oluşturulabilmiştir. Bu sayede, değer aralığı gibi belli bir algoritmadan bağımsız testlerin otomatik olarak oluşturulabilmesi sağlanmış, bu da önemli bir iş gücünün başka alanlara kaydırılabilmesine olanak yaratmıştır.

5. Sonuç

Bu makalede, gerçek zamanlı bir sistemin otomatik olarak test edilmesi yönünde bir fikir oluşturabilmek için ASELSAN MGEO grubuna bağlı Yazılım Doğrulama Ekibi tarafından geliştirilen test otomasyon altyapısından bahsedilmiştir. İzlenmiş olan yöntem ve karşılaşılan sorunlardan bahsedildikten sonra bu altyapının gerçek zamanlı bir sistemin testlerinde kullanılması neticesinde elde edilen kazanımlar paylaşılmıştır.

Bütün yazılım doğrulama süreçlerinde tekrar edilebilirlik önemli bir kıstastır. Aynı şekilde koşturulamayan bir test, hatanın anlaşılabilmesi için herhangi bir geri bildirim sağlayamayacaktır. Zamanlamanın kritik olduğu gerçek zamanlı sistem testlerinde, elle koşturulan iki testin birbiriyle tamamen aynı olduğunu söylemek neredeyse imkânsızdır.

Bunun yanı sıra, testlerin oluşturulması ve doğrulanması sırasında işgücü ve zamandan yapılan her türlü tasarruf, başka bir aktivitenin daha verimli bir şekilde gerçekleştirilmesi için bir fırsattır.

Bu durumlar göz önünde bulundurulduğunda, özellikle gerçek zamanlı sistemlerdeki testlerin otomatikleştirilmesinin ve bu otomatikleştirme işlemlerine yeni yaklaşımlar getirilmesinin önemi anlaşılabilmektedir.

6. Kaynaklar

[1] Software Test Automation: Myths and Facts, M.N. Alam, IMI Systems Inc.

[2] Why Automation Projects Fail (and what you can do to prevent failure), Art Beall, 2008

[3] Test Automation: From Record/Playback to Frameworks, John Kent M. Sc. http://www.simplytesting.com

[4] Fifth Generation Scriptless and Advanced Test Automation Technologies, Jeff Hinz, Martin Gijsen

[5] MIL-STD-1553 Tutorial, AIM GmbH, 2002

[6] When Should a Test Be Automated?, Brian Marick, 1998