Araştırma Makalesi
Adıyaman Üniversitesi Mühendislik Bilimleri Dergisi
21 (2023) 321-329

NESNEYE YÖNELİK PROGRAMLAMADA SINIF TEST ÜRETİMİ İÇİN OLAY TEMELLİ BİR YAKLAŞIM ÖNERİSİ

Tuğkan TUĞLULAR[1*]

[1]İzmir Yüksek Teknoloji Enstitüsü, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, İzmir, 35430, Türkiye
Geliş Tarihi/Received Date: 28.09.2023 Kabul Tarihi/Accepted Date: 30.12.2023 DOI: 10.54365/adyumbd.1368061

ÖZET

Nesneye yönelik programlamada yaşam döngüsü bulunan nesneleri temsil eden sınıfların test edilmesi önemlidir. Sınıf testleri, metot testleri ile hata bulunmadığı aşamada çalıştırılması gereken testlerdir. Metot testleri bir metodun beklendiği gibi çalıştığını doğrularken, sınıf testleri bir sınıfın farklı metot çağrıları karşısında geçerli bir durumda kaldığını ve beklendiği gibi yanıt verdiğini teyit eder. Metotlarının doğru çalışması bir sınıfın doğru çalıştığını göstermeyeceği için sınıf testleri önemlidir. Bu çalışmada sınıf testleri için olay temelli bir yaklaşım önerilmiştir. Bu öneri içinde formel Olay Sıra Çizgesi (OSÇ) kullanılmıştır. OSÇ'ler, Sonlu Durum Makineleri (SDM) üzerinden üretilebilen yönlü çizgeler olup çizge teorisinin algoritmalarını SDM'lerden farklı olarak doğrudan kullanabilmektedir. Bu avantajına ek olarak kodlama tarafında yine bu çalışmada önerilen şablonların kullanılması durumunda OSÇ'ler ile sınıf metotların eşleşmesi doğrudan ve hızlıca sağlanabilmektedir. OSÇ'ler için otomatik test üreten bir aracın varlığı ise bu çalışmayı doğrudan kullanılabilir kılmaktadır. Önerilen yaklaşım karmaşık bir yaşam döngüsüne sahip klima kontrol ünitesi sınıfı üzerinde doğrulanmıştır.

Anahtar Kelimeler: *Nesneye yönelik programlama, Sonlu durum makineleri, Olay sıra çizgeleri, Olay temelli test üretimi, Sınıf testleri*

PROPOSAL OF AN EVENT-BASED APPROACH FOR CLASS TEST GENERATION IN OBJECT-ORIENTED PROGRAMMING

ABSTRACT

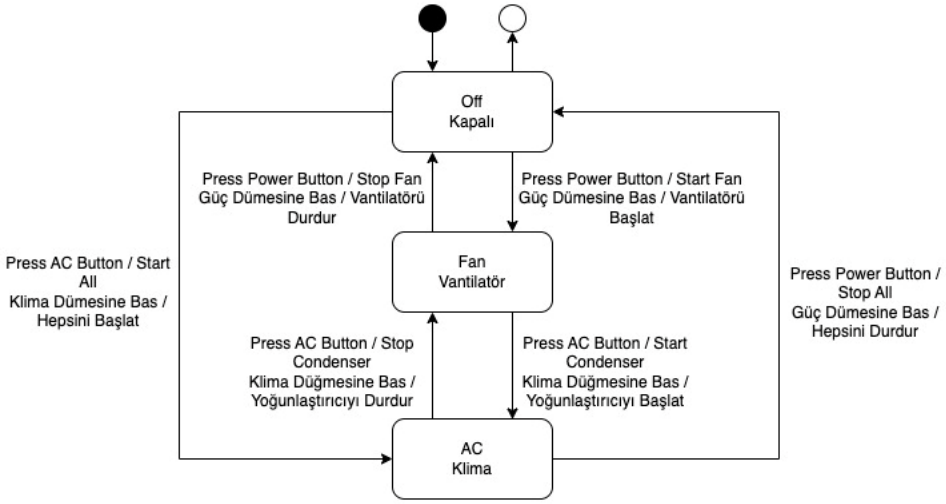
Testing classes with lifecycles is crucial in object-oriented programming. Class tests should be performed after method tests are error-free. Method tests check that a method operates as expected, while class tests verify that a class remains valid and reacts as expected throughout method calls. Class tests are crucial because method tests do not necessarily show class functionality. In this work, event-based class testing is proposed. Our concept uses formal Event Sequence Diagrams (ESGs). ESGs are directed graphs that may be created from Finite State Machines (FSMs) and use graph theory techniques directly. Additionally, using the templates provided in this study on the code side allows direct and fast mapping of ESGs and class methods. A tool that generates automatic ESG testing makes this study usable. The proposed method was validated on an air-conditioning control unit class with a complex lifecycle.

Keywords: *Object-oriented programming, Finite state machines, Event sequence graphs, Event-based test generation, Class tests*

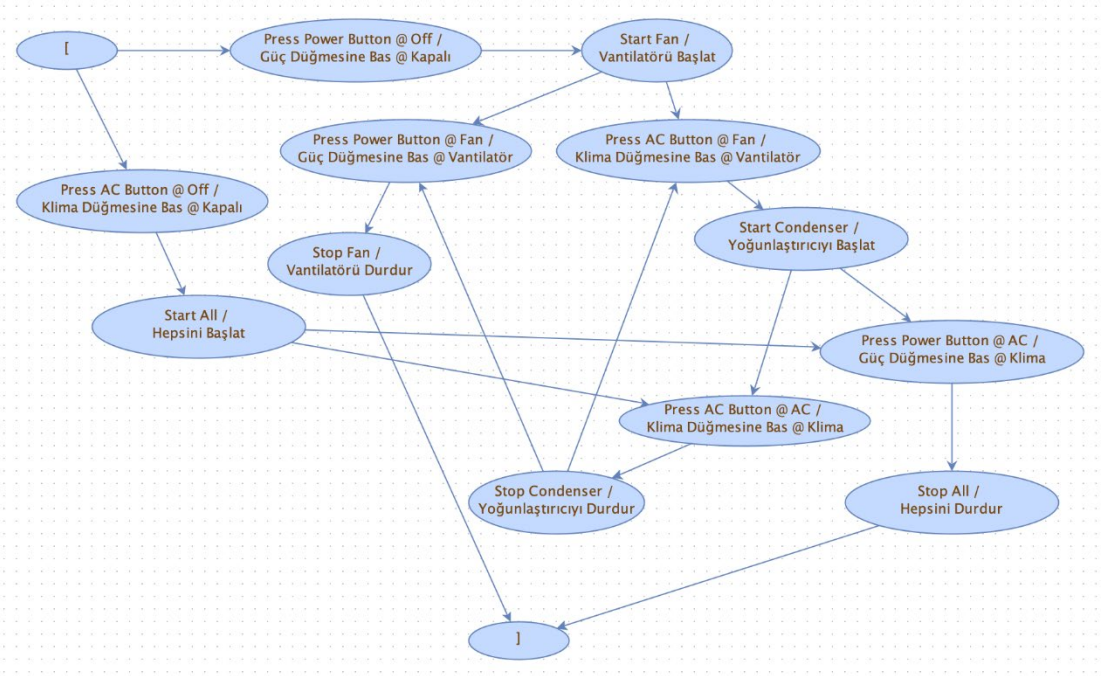
[*] e-posta[1] : tugkantuglular@iyte.edu.tr ORCID ID: https://orcid.org/0000-0001-6797-3913 (Sorumlu Yazar)

1. Giriş

Nesneye yönelik programlamada yaşam döngüsü bulunan nesnelerde bu yaşam döngüsü nesnenin oluşturulması ile başlar, genellikle kendisine gelen metot çağrıları ile devam eder, bu çağrılar nesnenin durumunu değiştirir ve nesne, kullanımı bittiğinde geri dönüşüme uğrar. Bu yaşam döngüsü, sonlu durum makinesi [1] veya olay sıra çizgesi [2] ile temsil edilebilir (bakınız Şekil 1 ve Şekil 2). Sonlu durum makinesi (SDM) bir nesnenin içinde bulunabileceği durumları ve bunlar arasındaki geçişleri temsil eder. Durumlar nesne yaşam döngüsünün çeşitli aşamalarını temsil ederken geçişler nesneyi bir durumdan diğerine taşıyan eylemleri/olayları temsil eder. Olay sıra çizgeleri (OSÇ) ise düğümlerin "takip eder" ilişkisini sağladığı kenarlarla birbirine bağlanan olayları temsil ettiği yönlü bir çizgedir. Bir SDM, durumları ve girdi/çıktıları birleştirerek bir OSÇ'ne dönüştürülebilir. OSÇ'ler anlamsal olarak zenginleştirilmiş Myhill çizgeleridir [3]. Hem sonlu durum makineleri hem de olay sıra çizgeleri olay temelli formel modelleme yöntemleridir [4].



Şekil 1. Klima kontrol ünitesi sonlu durum makinesi ([7] den değiştirilerek alınmıştır)



Şekil 2. Klima kontrol ünitesi olay sıra çizgesi

SDM ve OSÇ gibi formel modeller sayesinde model tabanlı sınama (test) mümkün olmaktadır. Model tabanlı sınamada modeller sınamanın iki ana problemini çözmektedir. İlk olarak formel modellerden sınama durumlarını otomatik üretmek için algoritmalar ve bu algoritmaların kullanıldığı yazılım araçları bulunmaktadır. İkinci olarak yazılım sınamadaki kahin problemini, bir sonraki aşamayı, ki bu bir durum ya da olay olabilir, doğal olarak içinde barındırdığı için ortadan kaldırmaktadır. Model tabanlı sınama ile üretilen sınama durumları işletilebilmek için test kodlarına dönüştürülmelidir. Ayrıca bunlar, metot (birim) testi, sınıf testi, bileşen testi, servis testi, uygulama/sistem testi ve kullanıcı kabul testi gibi farklı seviyelerde test kodlarına çevrilebilirler.

Nesneye Yönelik Programlama (NYP) birim testi uygulaması, bir sınıf içinde yer alan metotların çevredeki yazılımdan bağımsız olarak düzgün işlediğini doğrulamak amacıyla incelenmesidir. NYP bağlamında, sınıflar içindeki metotlar, nesnelerin sergilediği belirli davranışların uygulanmasıyla görevlidir. Nesneye yönelik programlamada bir sınıf işletildiğinde nesneye dönüşür [5]. Bu açıdan bakıldığında bir sınıf, nesnesinin taslağıdır. Bu taslak, nesneye yönelik programlamada, genellikle değişkenlerle ifade edilen özelliklerden ve işlevleri temsil eden metotlardan oluşur. Sınıf testi, sınıfın temsil ettiği nesnenin yaşam döngüsünü doğrulamayı amaçlar [6]. Bu bağlamda sınıf testi hiyerarşik olarak metot (birim) testinin hemen üstündedir. Birim testi, bir sınıf metodunun birlikte çalıştığı olduğu diğer sınıf ve metotlardan bağımsız olarak düzgün işlediğini doğrular. Sonraki aşama sınıf testidir. Bir nesnenin yaşam döngüsü içinde metotları hem nesne içinden hem de nesne dışından çağrılabilmektedir. Sınıf testinde bir nesnenin yaşam döngüsü içinde gelen metot çağrıları karşısında beklendiği gibi çalıştığını doğrulama için sınıf testi gerçekleştirilir. Bu çalışmada olay temelli OSÇ'ler kullanılarak sınıf testi için bir yaklaşım önerilmektedir. Bu makalede örnek uygulama olarak klima kontrol ünitesi sınıfı ve nesnesi kullanılacaktır. Java kodu İngilizce olduğu için şekillerde hem İngilizce hem de Türkçe karşılıkları verilmiştir.

Sonlu durum makineleri üzerinden sınama durumları veya test üretmek üzere geliştirilmiş yazılımlar bulunmaktadır. Ancak bunlar nesne güdümlü programlama yaklaşımı ile sorunsuz olarak örtüşmemektedir çünkü NYP'da metot çağrıları birer olaydır, durum değildir. Dolayısı ile bir diğer olay temelli modelleme yöntemi olan olay sıra çizgelerinin sınıf testi için daha uygun olduğu düşünülmektedir. Şekil 2'de klima kontrol ünitesi OSÇ'si gösterilmektedir. Bu OSÇ Şekil 1'de çizilmiş olan SDM'nin sadece olaylar temsil edilecek şekilde dönüştürülmesi ile elde edilmiştir. Bu şekilde sadece olaylardan ibaret yönlü bir çizge olmasından dolayı sınama durumlarını üretmek çizge teorisi algoritmalarını kullanarak mümkün olmaktadır [8]. Hatta bu algoritmalar kullanılarak Test Suite Designer (TSD) isimli bir yazılım aracı bilimsel çalışmalar için üretilmiştir [4]. Şekil 2 de bu TSD aracı kullanılarak çizilmiştir. TSD aracının en önemli özelliği OSÇ'lerden sınama ardışık sıralarını otomatik üretebilmesidir. Ancak TSD aracı bu çalışma ile ilk defa sınıf testi için kullanılacaktır.

Makale şu şekilde yapılandırılmıştır. Bölüm 2'de ilgili çalışmalar verilmiştir. Bölüm 3 makalede önerilen yaklaşımı detayları ile ortaya koymaktadır. Bölüm 4'te önerilen yaklaşım bir örnek üzerinde doğrulanmış ve önerilen yaklaşımın avantajları tartışılmıştır. Sonuç bölümü ile makale tamamlanmıştır.

2. İlgili Çalışmalar

Bu çalışmada kullanılan OSÇ'lere ilişkin tanımlar [1] ve [8] den aynen alınmıştır:

Tanım 1: Bir olay sıra çizgesi (OSÇ) (V, E) bir yönlü çizgedir; öyle ki $V \neq \emptyset$ sonlu bir olaylar (ya da düğümler) kümesi ve $E \subseteq V \times V$ sonlu bir kenarlar kümesidir. Olay sıra çizgelerinde "[" ve "]" sözde olayları sırasıyla başlangıç ve bitiş olaylarını işaretlemek için kullanılır [1].

Şekil 2'de görüldüğü üzere klima kontrol ünitesi OSÇ "[" düğümü ile başlamakta ve "]" düğümü ile bitmektedir. Her düğüm bir olayı temsil etmektedir. Yönlü kenarları tanımlayan olay ikilileri (İng. event-pairs) içinde ikinci olay ilk olayı izler diye yorumlanır.

Tanım 2: Tanım 1'de verilen V, E kümeleri ve OSÇ için herhangi bir ardışık olay dizisi 〈 $v_0, \ldots, v_k\rangle$ i=0, . . ., k-1 için $(v_i, v_{i+1}) \in E$ ise sıralı olay dizisi olarak adlandırılır [8].

Sıralı olay dizisi; olay ikilileri, olay üçlüleri vb. testin kapsama gücünü ifade etmek için kullanılır. Şekil 2'de verilen OSÇ için *〈Press Power Button @ Off / Güç Düğmesine Bas @ Kapalı, Start Fan / Vantilatörü Başlat〉*bir olay ikilisi, *〈Press Power Button @ Off / Güç Düğmesine Bas @ Kapalı, Start Fan / Vantilatörü Başlat, Press Power Button @ Fan / Güç Düğmesine Bas @ Vantilatör〉*bir olay üçlüsü olarak örneklendirilebilir. Bu çalışmada sadece olay ikilileri kapsaması dikkate alınmıştır, ancak kolayca olay üçlüleri ve olay dörtlüleri şeklinde genişletilebilir.

Tanım 3: Tam ardışık olay dizisi (İng. Complete Event Sequence) ise, OSÇ'nin başlangıç olayı ile başlar ve bitiş olayı ile sona erer [8].

Böylece bir nesneyi yaşam döngüsünün başından sonuna kadar test etmek mümkün olmaktadır. Eğer bir test için bir OSÇ'deki tüm olay ikilileri kapsanmak isteniyorsa tüm tam ardışık olay dizileri çıkarılmalıdır. Bunun için Chinese Postman Problem çözümü kullanılır [9]. Chinese Postman Problem çözümünün hedefi bir çizge için minimum uzunlukta kenar kaplama turunu bulmaktır. Bunun için geliştirilen algoritma kısaca bir güçlü bağlantılı çizgenin kenarlarını kapsayan minimal bir tur belirler. Bir ESG'nin güçlü bağlantılı bir grafiğe dönüştürülmesi yöntemi de [9]'da açıklanmıştır. Tüm tam ardışık olay dizilerini çıkarmak üzere bir araç geliştirilmiştir. Detayları [4]'den elde edilebilir. OSÇ orijinal olarak kullanıcı arayüzü testi için kullanılmıştır. Yazılım kullanıcı arayüzlerindeki etkileşim olaylar ile kolaylıkla açıklanabildiği ve modellenebildiği için bu alan OSÇ'ler yoğun olarak kullanılmıştır. Bu çalışmada ise OSÇ'ler sınıf testi için kullanılmıştır. Kullanıcı arayüzlerine benzer bir şekilde metot çağrılarının olaylar şeklinde ifade edilebilmesi ve modellenebilmesi OSÇ'leri sınıf testleri için kullanışlı kılmaktadır.

Sınıf testleri, metot (birim) testleri üzerine gerçekleştirilir. Metot testlerini oluşturma ve otomatikleştirme üzerine birçok çalışma sunulmuştur. Bunlardan biri sembolik test üretme yaklaşımıdır [10]. Bu çalışmada Giriş-Çıkış Sembolik Geçiş Sistemleri adı verilen, Giriş-Çıkış Otomatlarından [11] ve iletişim kuran ardışık süreçlerden [12] ilham alan bir genişletilmiş geçiş sistemi modeli kullanılarak test üretilmiştir. Benzer bir çalışmada Etiketli Geçiş Sistemleri'nin benzer şekilde test üretimi için kullanılabileceği gösterilmiştir [13].

NYP için durum temelli test çalışmaları 1970'lere kadar gitmektedir [14]. Bu çalışma her ne kadar metot bazlı birim testler için olsa da nesnenin durumu ile bağlantı kurulmuştur. Nesnenin durumunu dikkate alarak test üreten bir başka çalışmada test üretimi için yaklaşım ve araçlar ortaya konmuştur [15]. Sınıf testleri ile ilgili ilk çalışmalardan birinde UML Statechart kullanılarak durum temelli sınıf testleri üreten bir yöntem önerilmiştir [16]. Sınıf testleri ile ilgili en önemli çalışmalardan biri olan makalede, duruma bağlı bir davranış sergileyen sınıflar için iyi bilinen durum tabanlı test tekniklerinin maliyet etkinliği araştırılmıştır [6]. Bu makalede de nesne yönelimli programlama için sınıf test üretiminde sonlu durum makineleri kullanılması önerilmiştir.

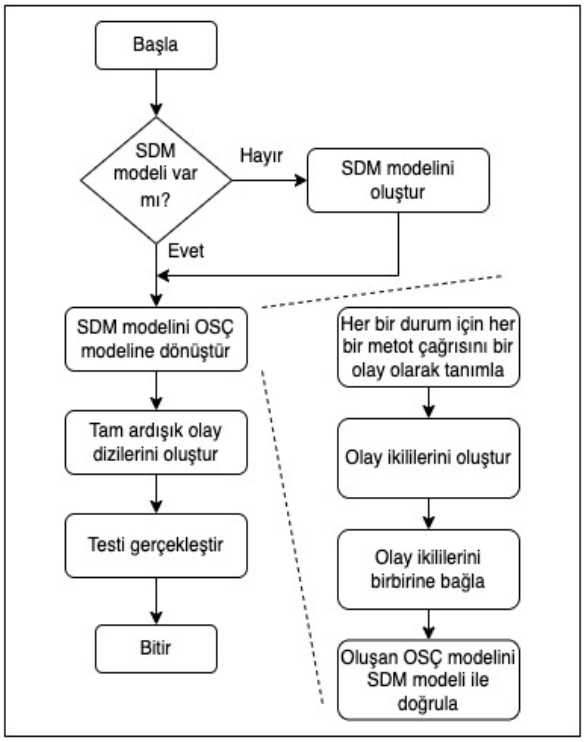
3. Olay Temelli Sınıf Test Üretimi için Bir Yaklaşım Önerisi

Çalışmada örnek olarak klima kontrol ünitesi sınıfı ve bu sınıftan üretilen nesne kullanılmıştır. Şekil 1'de de görüldüğü üzere klima kontrol ünitesi nesnesi üç durumdan birinde olacaktır: Kapalı (Off), Vantilatör (Fan) ve Klima (AC). Geçerli durum Kapalı ise ve güç düğmesine basılma olayı gerçekleşirse, vantilatör çalıştırılacak ve durum Vantilatör olarak değişecektir. SDM açısından bu şekilde ifade edilen iki durum arası geçiş Java sınıf kodunda güç düğmesine bas metodu ile gerçekleşmektedir. Formel model ile Java sınıf kodu arasındaki eşleme üretilen sınama durumlarının hızlıca ve kolayca çalıştırılması için önemli bir hale gelmektedir. Önerilen OSÇ temelli yaklaşım bunu sağlamaktadır. OSÇ içinde bulunan her düğüm klima kontrol ünitesi sınıfı içindeki bir metoda karşılık geldiği için klima

kontrol ünitesi nesnesinin yaşam döngüsünün olası akışları TSD aracı ile üretilebilmekte ve kolayca test koduna dönüştürülebilmektedir.

Geçerli durum Vantilatör ise ve güç düğmesine basılma olayı gerçekleşirse, vantilatör durdurulacak ve durum Kapalı olarak değişecektir. Geçerli durum Vantilatör ise ve klima düğmesine basılma olayı gerçekleşirse, yoğunlaştırıcı çalıştırılacak ve durum Klima olarak değişecektir. Geçerli durum Klima ise ve klima düğmesine basılma olayı gerçekleşirse, yoğunlaştırıcı durdurulacak ve durum Vantilatör olarak değişecektir. Geçerli durum Klima ise ve güç düğmesine basılma olayı gerçekleşirse, her şey kapatılacak ve durum Kapalı olarak değişecektir. Geçerli durum Kapalı ise ve klima düğmesine basılma olayı gerçekleşirse, her şey çalıştırılacak ve durum Klima olarak değişecektir.

Olay temelli sınıf test üretimi için SDM-OSÇ temelli yaklaşım Şekil 3'te verilmiştir. Yukarıda örnek üzerinde açıklanan yaklaşım burada bir akış diyagramı olarak verilmiştir. Eğer sınıf için bir SDM yok ise öncelikle bir SDM oluşturulur. Oluşturulan ya da var olan SDM, OSÇ'ne dönüştürülür. Bu dönüştürme işlemi, her bir durum için her bir metot çağrısını bir olay olarak tanımlayarak başlar. Ardından biri peşi sıra gelen olay ikilileri oluşturulur. Bu olay iklileri olaylar tekrar etmeyecek şekilde birbirine bağlanarak OSÇ modeli elde edilir. OSÇ modeli SDM modeline bakılarak doğrulanır.



Şekil 3. Olay temelli sınıf test üretimi için SDM-OSÇ temelli yaklaşım

Çalışmada kullanılan klima kontrol ünitesi örneği tipik bir nesne yaşam döngüsünü temsil etmektedir. Bu makalede klima kontrol ünitesi nesnesine ait yaşam döngüsü hem çok iyi bilinen SDM ile hem de çalışmada ifade edilen kolay ve hızlı test koduna dönüştürme farkını ortaya çıkaran OSÇ ile modellenmiştir. Klima kontrol ünitesi gibi yaşam döngüsüne sahip nesnelerin Java sınıf kodunun sayılabilir sonlu liste (İng. enumaration) yaklaşımı ile kodlanması gerçekleştirilmiştir. Klima kontrol ünitesi Java sınıfı çekirdeğinde durumlar ve olaylar için birer sayılabilir sonlu listesi barındırmaktadır. Java, sayılabilir sonlu listelerin içinde işlev ve veri üyelerinin bulunmasına izin verir. Böylece her olay karşında gerçekleştirilecek işlev ve değişecek veriler tanımlanmış olur. Klima kontrol ünitesi Java sınıfı çekirdeğini saran ve sınıf dışından çağrılara izin veren bir programlama arayüzü bulunmaktadır. Bu arayüz sayesinde klima kontrol ünitesi nesnesi hem kendi içinden hem de diğer nesnelerden çağrılar alabilmektedir. Bu çağrılara ilişkin yetki denetimi "public" ve "private" tanımları ile sağlanmaktadır.

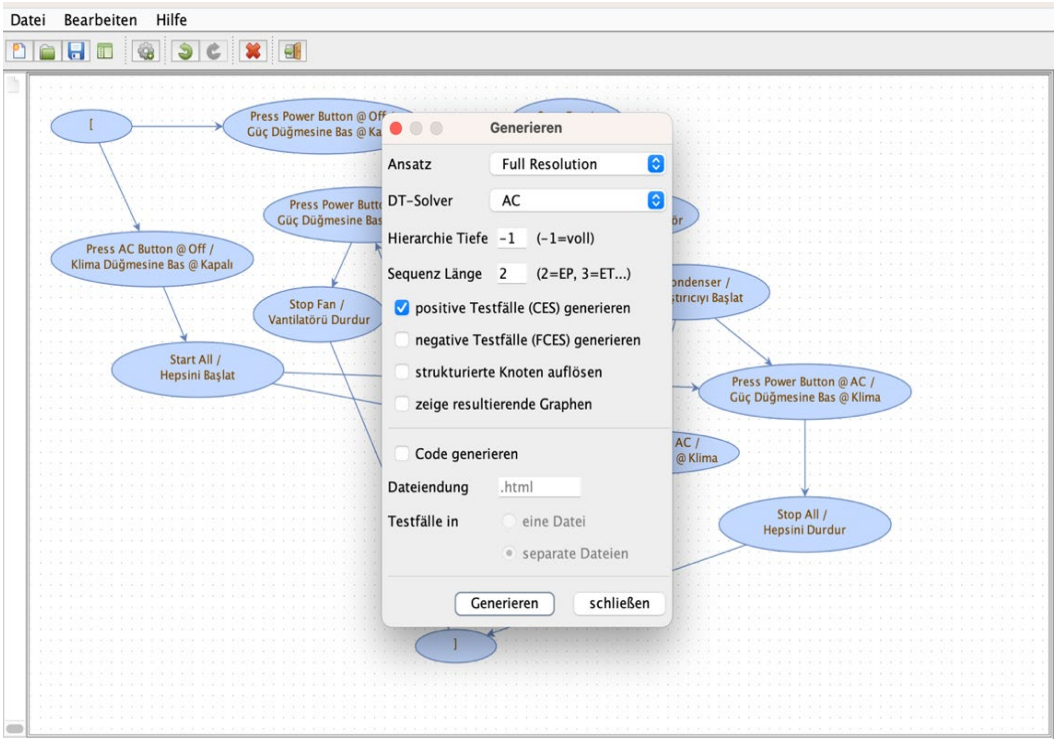
"Public" metotlar nesne dışından çağrılabilirken "private" metotlar sadece nesne içinden çağrılabilmektedir.

Bu bölümde şu ana kadar anlatılan yöntem makalede önerilen yaklaşımın bir parçası olup yaşam döngüsü olan sınıfların kodlanması için bir şablon ortaya koymaktadır. Bu şablonun özelliği olay temelli sınıf testleri üretme yaklaşımı ile birlikte çalıştığında test sürecini hızlandırması ve kolaylaştırmasının ötesinde bir hata bulunduğunda söz konusu hatanın hızlıca giderilmesini sağlıyor olmasıdır. Olay temelli sınama durumu üretimin ana avantajlarından birinin kahin problemini çözmesi olduğu yukarıda ifade edilmişti. Olay temelli sınamada kahin şöyle çalışmaktadır. OSÇ ile ortaya çıkarılan tam ardışık olay dizileri hem girdi olaylarını (örneğin, güç düğmesine basılması) hem çıktı olaylarını (örneğin, fanın başlatılması) içerdiği için bir girdi olayı sonucunda takip eden çıktı olayı gerçekleşmez ise bu hataya işaret eder. Böylece hatanın hangi noktada ortaya çıktığı da bilindiği için hata bulunma, giderme ve tekrar test etme işlemleri hızlıca gerçekleştirilebilir.

Bu çalışmada OSÇ düğümü içine durum bilgisini @ etiketi ile katarak yeni bir çözüm üretilmiştir. Bu yaklaşımın amacı doğru olay sıraları elde edilmesidir. Söz konusu @ etiketi sayesinde SDM'den OSÇ'ne çevirim işlemi de daha kolay olmaktadır. Bu makalede önerilen yaklaşımın bir diğer avantajı ise OSÇ'lerde sadece olaylar olduğu için OSÇ'ler ile olay patikalarının daha iyi gözlenmesidir. Bir nesnenin işleyişi de metot çağrıları sayesinde gerçekleştiği için ve her metot da bir olay olarak modellenebildiği için olay temelli model ile kod arasında net ve kolay bir eşleme sağlanmaktadır. Bunun bir örneği takip eden bölümde verilmiştir.

4. Değerlendirme

Bu makalede önerilen yaklaşımın doğrulaması (İng. validation) için yaklaşımın üzerinden anlatıldığı örnek olan klima kontrol ünitesi kullanılmıştır. Değerlendirme noktasında yukarıda ifade edilen şekilde klima kontrol ünitesi için OSÇ'nin ve Java sınıf kodunun hazır olduğu kabul edilmiştir. Ardından TSD aracı ile Şekil 4'te görüldüğü gibi olay ikilileri için tam ardışık olay dizileri otomatik olarak oluşturulmuştur.



Şekil 4. TSD aracı ile olay ikilileri için tam ardışık olay dizileri üretilmesi

TSD aracı tarafından üretilen 14 düğüm ve 18 yönlü kenar içeren klima kontrol ünitesi OSÇ için olay ikilileri için üretilen dört tam ardışık olay dizileri şöyle listelenebilir:

4: [, Press Power Button @ Off / Güç Düğmesine Bas @ Kapalı, Start Fan / Vantilatörü Başlat, Press Power Button @ Fan / Güç Düğmesine Bas @ Vantilatör, Stop Fan / Vantilatörü Durdur,],

6: [, Press AC Button @ Off / Klima Düğmesine Bas @ Kapalı, Start All / Hepsini Başlat, Press AC Button @ AC / Klima Düğmesine Bas @ Klima, Stop Condenser / Yoğunlaştırıcıyı Durdur, Press Power Button @ Fan / Güç Düğmesine Bas @ Vantilatör, Stop Fan / Vantilatörü Durdur,],

4: [, Press AC Button @ Off / Klima Düğmesine Bas @ Kapalı, Start All / Hepsini Başlat, Press Power Button @ AC / Güç Düğmesine Bas @ Klima, Stop All / Hepsini Durdur,],

10: [, Press Power Button @ Off / Güç Düğmesine Bas @ Kapalı, Start Fan / Vantilatörü Başlat, Press AC Button @ Fan / Klima Düğmesine Bas @ Vantilatör, Start Condenser / Yoğunlaştırıcıyı Başlat, Press AC Button @ AC / Klima Düğmesine Bas @ Klima, Stop Condenser / Yoğunlaştırıcıyı Durdur, Press AC Button @ Fan / Klima Düğmesine Bas @ Vantilatör, Start Condenser / Yoğunlaştırıcıyı Başlat, Press Power Button @ AC / Güç Düğmesine Bas @ Klima, Stop All / Hepsini Durdur,],

Formel ve teorik yaklaşımın avantajı burada görülmektedir. Toplam olay sayısı asgariye indirilecek şekilde tüm olay ikililerini kapsayacak tam ardışık olay dizileri otomatik olarak üretilmektedir. Kişiler test üretme sürecinden soyutlandığı için hata olasılığı olmayacak derecede azaltılmakta ve çok hızlı testler üretilmektedir. Yukarıda listelenen tam ardışık olay dizileri milisaniye düzeyinde oluşturulmaktadır. Ayrıca, toplam olay sayısı asgariye indirildiği için test süreci de kısalmaktadır. Bu avantajları açısından OSÇ'lerin sınıf testi için kullanılması önerilmektedir.

Dört uzunluğundaki ilk sınama ardışık durumu için test kodu aşağıdaki örnekte görüldüğü gibi hızlıca üretilebilir ve bir test sınıfı içinde çalıştırılabilir:

```
pressPowerButton(state.Off);
checkStartFan();
pressPowerButton(state.Fan);
checkStopFan();
```

Burada dikkat edilmesi gereken iki konu vardır. Birincisi, klima kontrol ünitesi Java sınıfında argüman almayan pressPowerButton() metodu bulunmakla beraber durumu (örneğin, state.Off) argüman olarak alan bir pressPowerButton() metodu bulunmamasıdır. Bu nedenle argüman almayan pressPowerButton() metodunu saran durumu argüman olarak alan bir pressPowerButton() metodu ayrıca kodlanmalıdır. Saran metot, argüman olarak gelen durumu klima kontrol ünitesi nesnesinin durumu ile kontrol etmeli ve gelen durum ile geçerli durum aynı değil ise istisna (İng. exception) vererek çıkmalıdır. Böylece bir sorun ya da hata olduğu anlaşılacaktır. İkinci konu ise, vantilatörün çalıştırılması bir çıktı olayı olduğu için bu olayı gözleyecek "public" bir metot oluşturulması gereklidir. NYP için önemli bir özellik olan "private" metotların nesne dışından çağrılmasının mümkün olmadığını yukarıda belirtmiştik. Oysa test sınıfının oluşturduğu test nesnesi sınanan nesnenin dışında bir nesne olduğu için "private" metotları çağıramayacaktır. Bu nedenle gözlem metotların "public" olarak sadece test amaçlı olsa bile hazırlanması önemlidir. Bu gözlem metotlarının gözlem beklendiği gibi ise sadece dönmesi yeterli iken gözlem beklendiği gibi değil ise istisna vererek sonlanması gereklidir. Böylece bir sorun ya da hata olduğu anlaşılacaktır.

Metot (birim) testinde olduğu gibi test edilen metodu izole etmek için ekstra kod yazılması gerektiği gibi sınıf testi için de yukarıda açıklandığı şekilde ekstra kod yazılması gerekmektedir. Her ne kadar bu kodların şablonu sabit ve içindeki kodlar basit de olsa zaman alacaktır. Bu zaman kaybını asgariye indirmek için kod üreteçleri kullanılabilir. Söz konusu kod üreteçlerinin verimli olması için yazılımın yukarıda belirtilen şablonlar kullanılarak geliştirilmesi gereklidir.

Bu makalede önerilen yaklaşım ile ilgili dikkat edilmesi gereken bir diğer konu ise her sınıf için bu yaklaşımın çalışmayabileceğidir. Makalenin başında ifade edildiği gibi bu yaklaşım bir yaşam döngüsü olan nesneler içindir. Ancak NYP'da, örneğin görevi sadece veri tutmak olan veri nesneleri de bulunmaktadır. Bu tür nesneler için bu çalışmada önerilen yaklaşım geçerli değildir.

5. Sonuçlar

Nesneye yönelik programlamada, yaşam döngüleri olan nesneleri tanımlayan sınıfların test edilmesi gerekir. Metot testlerinde hiçbir hata bulunmadığında çalıştırılması gereken sınıf testleri, hiyerarşik olarak metot testlerinin üstündedir. Metot testleri, bir metodu girdi olarak ne aldığı ve çıktı olarak ne verdiği açısından değerlendirerek metodun beklendiği gibi çalışıp çalışmadığını doğrular. Öte yandan sınıf testleri, bir sınıfın geçerli bir durumda kalmasını ve farklı yöntemler çağrıldığında bile beklendiği gibi tepki vermesini doğrular. Sınıf testleri önemlidir çünkü kendisine ait metotların doğru çalışması sınıfın doğru çalıştığı anlamına gelmez. Bu çalışmada yaşam döngüsü olan nesneleri temsil eden sınıflar için olaylara dayalı bir test yöntemi önerilmektedir. Bu öneride Olay Sıra Çizgelerinden (OSÇ) yararlanılmıştır. OSÇ'ler, Sonlu Durum Makinelerinden (SDM) oluşturulabilen yönlü çizgelerdir ve SDM'lerin aksine, doğrudan çizge teorisi yöntemlerini kullanabilmektedir. Bu faydanın yanı sıra bu çalışmada önerilen şablonların kodlama için kullanılması durumunda OSÇ'ler ile sınıf metotlarının eşleştirilmesi kolay ve hızlı olacaktır. OSÇ'lerden otomatik olarak ve hızlı test üretebilen bir araç olması sayesinde bu çalışma hemen kullanılabilir. Önerilen yöntemi doğrulamak için karmaşık bir yaşam döngüsüne sahip bir klima kontrol ünitesi sınıfı kullanılmıştır.

Gelecek çalışma olarak iki yön belirlenmiştir. İlk olarak önerilen yaklaşımın mutasyon analizi ile değerlendirilmesi yapılacaktır. Metotlar mutasyona uğratılacak ve sınıf testlerinin bunları yakalama özelliği araştırılacaktır. İkinci yön olarak, sınıf testi için önerilen yaklaşım bileşen (İng. component) testi için uyarlanacaktır. Bu uyarlama sırasında bileşeni oluşturan sınıfların testlerinin bileşen testi için tekrar kullanımı değerlendirilecektir.

Kaynaklar

[1] Wagner F, Schmuki R, Wagner T, Wolstenholme P. Modeling software with finite state machines: a practical approach. CRC Press, 2006.
[2] Belli F. Finite state testing and analysis of graphical user interfaces. Proceedings 12th International Symposium on Software Reliability Engineering, Nov. 2001; 34–43.
[3] Myhill J. Finite automata and the representation of events. WADD Technical Report 57. 1957: 112–137.
[4] Belli F, Linschulte M, Tuğlular T. Karar tablosu destekli olay sıra çizgeleri temelli sınama durum üretim aracı. 10th Turkish National Software Engineering Symposium (UYMS 2016). CEUR Workshop Proceedings, 2016.
[5] Holland IM, Lieberherr KJ. Object-oriented design. ACM Computing Surveys (CSUR). 1996; 28(1): 273-275.
[6] Briand LC, Di Penta M, Labiche Y. Assessing and improving state-based class testing: A series of experiments. IEEE Transactions on Software engineering 2004; 30(11): 770-783.
[7] Ali J. Using Java Enums to implement concurrent-hierarchical state machines. Journal of Software Engineering. 2010; 4(3): 215-30.
[8] Belli F, Budnik CJ. Minimal Spanning Set for Coverage Testing of Interactive Systems. Theoretical Aspects of Computing - ICTAC 2004, Guiyang, China, 2004; 220–234.
[9] Belli F, Budnik CJ. Test minimization for human-computer interaction. Applied Intelligence 2007; 26: 161-174.
[10] Rusu V, Du Bousquet L, Jéron T. An approach to symbolic test generation. Proceedings of Integrated Formal Methods: Second International Conference (IFM 2000) Germany, Nov. 1–3, 2000; 338-357.
[11] Lynch N, Tuttle M. An introduction to input/output automata. CWI Quarterly 1999; 3(2).
[12] Hoare CAR. Communicating sequential processes. Prentice Hall International Series in Computer Science, 1985.

[13] Frantzen L, Tretmans J, Willemse TA. A symbolic framework for model-based testing. Proceedings of International Workshop on Formal Approaches to Software Testing, Aug 15, 2006; 40-54.
[14] Reed HG, Turner CD, Aibel JB, Dalton JT. Practical object-oriented state-based unit testing. WIT Transactions on Information and Communication Technologies, 1970; 9.
[15] Turner CD, Robson DJ. The state-based testing of object-oriented programs. Proceedings of IEEE Conference on Software Maintenance, Sep 27, 1993; 302-310.
[16] Tsai BY, Stobart S, Parrington N. A Method for Automatic Class Testing (MACT) Object-Oriented Programs Using A State-based Testing Method. Proceedings of 5th European Conference Software Testing Analysis & Review, EuroSTAR, Nov. 1997; 403-415.