Koşutzamanlı Yazılım Bileşenleri için Bir Otomatik Doğrulama Çerçevesi: VyrdMC

Tayfun ELMAS[1] **Serdar TAŞIRAN**[2]

[1, 2] Koç Üniversitesi, Bilgisayar Mühendisliği Bölümü, 34450, Sarıyer-İstanbul

[1]e-posta: telmas@ku.edu.tr [2]e-posta: stasiran@ku.edu.tr

Özet

Bu bildiri koşutzamanlı yazılım bileşenleri için, yapısal test, model denetleme ve çalışma zamanı arıtma (refinement) denetleme aracımız olan Vyrd'i birleştiren, VyrdMC adında bir çalışma zamanında doğrulama çerçevesi sunmaktadır. Yürütme tabanlı bir model denetleyici her bir test senaryosu için tüm iş parçacıklarının farklı zamanlamaları ile oluşan tüm yürütmeleri oluştururken, Vyrd yürütmeleri arıtma ihlallerini tespit etmek için izler. Bu bütünleşik yaklaşım, çalışma zamanı denetlemenin kapsamasını artırma avantajına sahiptir. Bir yararı da Vyrd'in model denetleyicinin çalışma zamanı ortamını ve kod donatma mekanizmasını, yürütmeleri arıtma denetleme sırasında tekrar oluşturmak için kullanmasıdır; bu yöntem kodun elle düzenlenmesini gereğini ortadan kaldırır. Bildiride çerçeveyi yazılım geliştirme süreci boyunca kullanmanın avantajları da tartışılmaktadır.

Abstract

This paper presents VyrdMC, a runtime verification framework for concurrent software components that combines structured testing, model checking, and our runtime refinement checker Vyrd. An execution-based model checker explores for each test case all distinct thread interleavings while Vyrd monitors executions for refinement violations. This combined approach has the advantage of improving the coverage of runtime refinement checking. As a side benefit, Vyrd reuses the model checker's runtime environment and instrumentation mechanism for replaying executions for refinement checking which reduces the need for manual annotation. We discuss benefits of using our framework throughout the software development process.

1. Giriş

Koşutzamanlı bileşenler pek çok endüstriyel ölçekte yazılım uygulamasında yoğun olarak kullanılmaktadır. Bu ölçekteki yazılımların doğrulanmasının zaman ve kaynak maliyeti, oluşabilecek durum uzayının (*state-space*) büyüklüğü ve iş parçacıklarının (*thread*) farklı şekillerde zamanlamaları (*scheduling*) sonucu ortaya çıkan çok sayıda yürütme nedeniyle yüksektir. Bu bildiri, koşutzamanlı yazılımlar için çalışma zamanı doğrulama (*runtime verification*) ve model denetleme (*model checking*) yöntemlerini birleştiren bir doğrulama çerçevesi sunmaktadır.

Bileşen tabanlı bir yazılımda bileşenler çevreleri tarafından genel (*public*) metotları çağrılarak erişilir. Koşutzamanlı bir ortamda her iş parçacığı yapılan işlemin doğruluğunu yargılamak için kullandığı bir bileşenin genel metotlarının görevini "atomik" olarak yerine getirdiğini varsayar. Atomik çalışan metotlar çağıran iş parçacığına, metot boyunca bileşenin araya başka bir iş parçacığına ait işlem girmeden gözlemlendiği/güncellendiği izlenimi verir. Metotların atomik

çalışmaları, performansı artırmak amacıyla kod gövdeleri tamamen senkronize edilmeden, yalnızca metot boyunca yerel senkronizasyonlarla sağlanabilmektedir. Bu şekilde gerçekleştirilen metotların atomik olma özellikleri indirgeme (*reduction*) [1] ve saflık (*purity*) [2] konusunda yapılan önceki çalışmalarla incelenmiştir. İndirgeme ve saflık, metodun bu özelliği denetlerken metot boyunca ulaşılan tüm değişkenlerin atomik olarak erişilmesini şart koşar. Ancak, bu şartı sağlamayan fakat doğru çalışan bazı atomik metot gerçekleştirimleri için yanlış uyarılar vermektedir. Taşıran ve Qadeer "arıtmayı (*refinement*)" koşutzamanlı bileşenler için bir doğruluk kriteri olarak önermiştir [3]. Arıtma genel olarak yazılım ve donanım sistemlerinin daha soyut bir şekilde betimlenmiş modellerinin gerçekleştirime doğru olarak aktarılıp aktarılmadığını kontrol etmek için tanımlanmıştır. Yazılım için arıtma bir doğruluk kriteri olarak, bileşenin gerçekleştiriminin (*implementation*) koşutzamanlı bir çevredeki her yürütmesine karşılık bileşenin atomik belirtiminin (*specification*) bir yürütmesi olmasını şart koşar.

Arıtmayı denetlemek için yürütme-tabanlı model denetleme yöntemleri endüstriyel ölçekte karmaşık bileşenler için durum uzayının çok büyük olması (*state space explosion* problemi) ve iş parçacıklarının farklı zamanlamaları sonucu çok sayıda farklı yürütme oluşabilmesi nedeniyle sınırlı şekilde uygulanabilmektedir. Yalın test yöntemi bu düzeyde karmaşık yürütme uzayı olan yazılımlar için yeterli kapsama (*coverage*) sağlayamadığı gibi çoğu durumda test süresince elde edilen kapsamanın miktarı da ölçülememektedir. Çalışma zamanı doğrulama yaklaşımı yürütmenin testten daha iyi gözlenmesini ve böylece arıtma gibi daha kapsamlı kriterlerin sınanabilmesini sağlar. Çalışma zamanı doğrulama, elde edilen kapsama miktarı bakımından test bazlı yaklaşımlar ile etraflı doğrulama (*exhaustive verification*) yöntemlerinin arasında yer almasına rağmen, endüstriyel ölçekte bileşenlere etkin olarak uygulanabilmektedir. Ancak bu yöntem denetlenecek yürütmeleri oluşturmak için bileşen üzerinde rasgele metot çağrıları yapan test programları kullandığı için testin getirdiği yetersiz kapsama problemiyle karşı karşıya kalmaktadır.

Bu bildiride testin kapsamasını artırmak için model denetleyici kullanan bir yaklaşım sunulacaktır. Bu yaklaşım, yapısal bir test yöntemi, model denetleyici ve daha önce geliştirilen arıtma denetleme aracı Vyrd'i [4] içerisinde birleştiren bir çerçeveye entegre edilmiştir. "VyrdMC" adını alan çerçeve [5], bileşenin koşutzaman karakteristiklerini taşıyan yürütme izleri (*execution trace*) oluşturmak için yapısal test programları kullanmaktadır. Test birimi tarafından üretilen her bir test senaryosu, iş parçacıklarının zamanlaması nedeniyle farklı davranışlar sergileyeceği için testin çalışması model denetleyici tarafından kontrol edilmekte ve test senaryosunun her çalışması için farklı bir yürütme elde edilmektedir. Bunun sonucu olarak da kapsamada sürekli bir artış sağlanmaktadır. Vyrd, model denetleyici ile paralel olarak çalışarak üretilen yürütme izlerini arıtma kriterlerine göre denetlemektedir.

Testin çalışmasının kontrolü yanında model denetleyicinin çalışma zamanı ortamı (*runtime environment*) Vyrd'in gereksinim duyduğu bazı görevlerin otomatikleştirilmesi amacıyla kullanılmaktadır.

Bu görevler (1) test sırasında oluşan çalışmanın izlenerek ilgili yürütme izinin Vyrd tarafından analiz edilecek biçimde çıkartılması, ve (2) Vyrd'in yürütme izini testin çalışmasını etkilemeden analiz etmesi için yürütmedeki işlemlerin farklı bileşen örnekleri üzerinde tekrar uygulanmasıdır.

Bu bildiride VyrdMC'nin yazılım geliştirme sürecinin birim test evresine entegre edilmesi ve böylece koşutzamanlı bileşenlerden oluşan uygulamaların koşutzamanlı ortamın getirdiği sorunların geliştirme aşamasında sürekli olarak denetlenmesi önerilmektedir. Bu bildiride anlatılan yöntem ve bu yöntemi otomatikleştirmek için yapılan çalışmalar bu entegrasyonu programcı açısından

kolaylaştırmayı hedeflemektedir. Bölüm 2'de doğrulama sürecine girecek bileşenlerin özelliklerini ve bu bileşenler için arıtma kriterlerini tanıtılmaktadır. Bölüm 3'de VyrdMC çerçevesi, içerdiği birimler ve Vyrd kullanılarak elde edilen deneyimlerle birlikte ayrıntılı olarak anlatılmaktadır. Bölüm 4'de çerçevenin yazılım sürecine entegre edilmesinin önemi vurgulanmıştır. Bildiri, sonuçların sunulduğu Bölüm 5 ile sona ermektedir.

2. Koşutzamanlı Yazılım Bileşenleri ve Arıtma Denetimi

Koşutzamanlı bir bileşen nesneye yönelik bir programlama dilinde yazılan bir sınıf tarafından temsil edilir. Bu bildiride sözü edilecek bileşenler çevrelerini oluşturan iş parçacıkları tarafından bileşenin genel metotları çağrılarak güncellenir ya da sorgulanır. Bu durumda bir bileşen ve çevresi arasındaki veri alışverişi metot argümanları ve dönüş değerleri ile gerçekleşir.

Bileşenin çevresini oluşturan tüm yazılım birimleri, bileşenin metotlarının atomik olarak çalıştığını varsayar. Bu durumda bileşenlerin tümünden oluşan programın doğruluğu bileşenin çevresine sunduğu atomik arayüzü doğru olarak gerçekleştirmesine bağlıdır. Doğru gerçekleştirime sahip bir bileşen koşutzamanlı bir ortamda çevresine, aynı anda sadece bir iş parçacığı bileşenin bir metodunu çalıştırıyor izlenimi verir.

Bu bildiri arıtmayı koşutzamanlı bileşen gerçekleştirimleri için bir doğruluk kriteri olarak sunmaktadır. Arıtma metotlara yerel olarak yerleştirilen iddialardan (*assertion*) daha kapsamlı bir kriter tanımlarken, yürütmeler test bazlı yöntemlerden daha açık bir şekilde gözlenerek test gibi sadece girdi çıktı bilgisine dayanarak denetleme yapan yöntemlerce bulunamayan hataların tespit edilmesini sağlar. Gerçekleştirimde arıtma ihlalleri senkronizasyon hataları [12] sonucu ortaya çıkar. Çoğu senkronizasyon hatası veri kaybı ya da bozulmasına neden olduğu için veri merkezli uygulamalar için arıtmayı önemsenmesi gereken bir kriter haline getirmektedir.

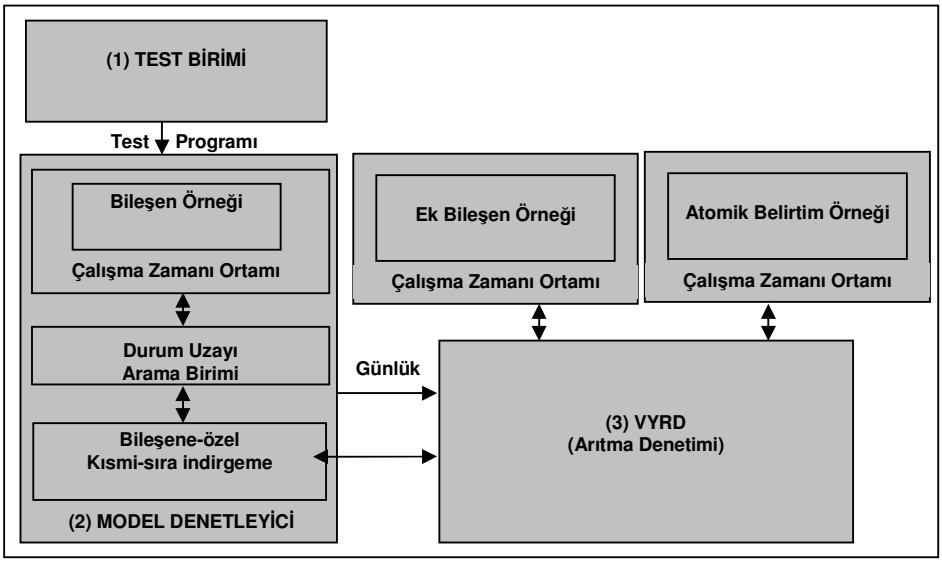
İndirgeme ve saflıktan farklı olarak arıtma, atomik çalışma kriterini metotların tüm değişkenlerin bir alt kümesi üzerindeki işlemleri için tanımlar. Arıtma, gerçekleştirime ait her yürütme izine karşılık atomik belirtime[1] ait bir yürütme izi var olmasını şart koşar. Gerçekleştirime ve belirtime ait yürütme izleri arasındaki ilişki (eşdeğer olup olmadıkları), Taşıran ve Qadeer [3] tarafından iki farklı arıtma yaklaşımıyla ortaya konulmuştur: girdi/çıktı-arıtma ve görünüm-arıtma. Girdi/çıktı-arıtma (*I/O-refinement*), izler boyunca uyuşan metot çağrıları için dönüş değerlerini denetlerken, görünüm-arıtma (*view-refinement*) bileşenin yürütme izi boyunca belirli noktalarda gerçekleştirim ve belirtimin durum bilgileri arasında bir denklik ilişkisi sağlanmasını şart koşar. Görünüm-arıtma bu özelliği ile içsel yapıda ortaya çıkan fakat metot dönüş değerlerine yansımayan hataları tespit edebilmektedir. Takip eden bölümlerde anlatılacak olan doğrulama çerçevesi koşutzamanlı bileşen gerçekleştirimlerini her iki arıtma kriteri için denetleyebilmektedir.

3. VyrdMC Doğrulama Çerçevesi

Bu bölümde koşutzamanlı bileşenleri yukarıda tanıtılan arıtma kriterlerine göre denetlemek için oluşturulan VyrdMC çerçevesinin ana bileşenleri tanıtılacaktır. Bu bildiride anlatılan mekanizmaların ayrıntılı çalışma prensipleri için [4] ve [5]'e başvurulabilir. VyrdMC'nin mimarisi Şekil 1'de gösterilmiştir. VyrdMC başlıca üç bileşenden oluşmaktadır: (1) bileşeni sınamak amaçlı

[1] Bu atomik belirtim için herhangi bir biçimde tanımlanabileceği gibi özgün bileşenin tüm metotlarının senkronize edilmesiyle oluşturulan bir atomik versiyon da belirtim olarak kullanılabilmektedir.

test programları üreten bir test birimi, (2) üretilen test programlarını farklı yürütmeleri oluşturacak şekilde çalıştıran bir model denetleyici, (3) model denetleyici tarafından oluşturulan yürütme izlerini arıtma kriterlerine göre denetleyen Vyrd.



Şekil 1. VyrdMC'nin mimarisi.

3.1 Test Birimi ve Test Senaryoları

Test biriminin ana amacı, doğrulama süreci boyunca ihtiyaç duyulacak kaynakları en aza indirecek şekilde eniyilenmiş test senaryoları oluşturmaktır. Bir test senaryosu, bileşenin metotlarını koşutzamanlı ortamda hataya odaklı çalıştıran bir programdır. Çalışma ortamının koşutzamanlı olması nedeniyle test senaryosu, girdileri aynı olsa bile farklı çalışmalarda iş parçacıklarının zamanlamalarındaki farklılıklar nedeniyle önceki çalışmalarda gözlenemeyen farklı davranışlar sergileyebilir. Bunun sonucu olarak, hatanın oluşmasını sağlayabilecek bir test senaryosu, birçok kez çalıştırılmasına karşın hata için gerekli iş parçacığı zamanlamasının sağlanamaması nedeniyle hatalı durumu sergilemeyebilir. Bununla birlikte bir çalışmada oluşan hata, sonraki çalışmalarda ortaya çıkmayabilir.

Bir test senaryosuna ait tüm koşutzamanlı yürütmelerin sayısı düşünüldüğünde hataya eğimli davranışlar üzerine odaklanmış test senaryoları üretebilmenin ve bu senaryoları hatayı oluşturabilecek şekilde çalıştırabilmenin önemi ortaya çıkmaktadır: Çoğu senkronizasyon hatası, girdi olarak verilen yeterince karmaşık yapıdaki bir bileşen örneğinin sınırlı bir parçası üzerinde eşzamanlı çalışan az sayıda bireysel metot tarafından gerçeklenebilmektedir. Bu göz önünde bulundurularak her test senaryosu, koşutzamanlı çalışmaları test edilecek metotların bir altkümesini programcıdan girdi olarak alır. Test birimi verilen her metot için o metodu çalıştıracak bir iş parçacığı yaratır. Metotlara aktarılacak argümanların değerleri metotların bileşenin içsel veri yapısında aynı kısımlar üzerinde çalışmasına imkan verecek şekilde birbiriyle korelasyon oluşturan

değerler arasından seçilir. Böylece veri bağımlılığı olan kod parçalarının oluşturacağı senkronizasyon hatalarına odaklanılmış olur.

Bellek kullanımı ve doğrulama süresinin metot sayısı ve dolayısıyla iş parçacığı sayısı ile orantılı olarak artacağı göz önünde bulundurularak, sınamaya yalnızca iki metodun çalıştığı senaryolardan başlanarak en bariz senkronizasyon hatalarının yakalanması sağlanır. Doğrulama işleminin ilerleyen safhalarında metot sayısı -ve dolayısıyla da iş parçacığı sayısı- artırılarak daha karmaşık davranışlar sonucu ortaya çıkacak hatalar hedeflenir.

Bu bildiride önerilen yöntem, bileşenin her yürütmesini, her biri atomik olarak çalışan ve "eylem" adı verilen kod parçalarının ardışık sıralanmış hali olarak görür. Test süresince çalıştırılan her eylem bilgisi, doğrulama sırasında testin koşutzaman karakteristiğini etkilememek için bütün-sıralı olarak paylaşımlı bir günlüğe kaydedilir. Doğrulama işlemi test ile eşzamanlı olarak bu günlükten okuyarak çalışır.

3.2 Kapsamanın Artırılması için Model Denetleyici Kullanılması

Bir test senaryosunda sınanan metotlar ve girilen argümanlar aynı bile olsa, senaryonun her çalışması için iş parçacıklarının zamanlamasına bağlı olarak farklı yürütmeler oluşabilmektedir. İş parçacıklarının zamanlamasının rasgele gerçekleştiği testlerde kapsamanın düşük olması problemi ile karşılaşılmaktadır. Ayrıca önceki test çalışmaları sonucu elde edilen toplam kapsama ölçülememektedir. Özellikle test altyapısı üzerine kurulan çalışma zamanı doğrulama yöntemleri için kapsamanın kontrolü ve ölçülmesi büyük önem taşımaktadır.

Bu bildiride kapsamanın kontrolü amacıyla test programlarının bir model denetleyici tarafından çalıştırılması önerilmektedir. Model denetleyici, verilen bir programı kendi çalışma zamanı ortamında kontrollü bir şekilde çalıştırarak programın farklı tüm yürütmelerinin ilgili analiz birimi tarafından izlenmesini sağlar [11]. Bu analiz birimi model denetleyicinin içine entegre edilebileceği gibi, VyrdMC'de olduğu gibi model denetleyicinin ürettiği yürütme izini ayrı olarak inceleyen bağımsız bir birim de (örneğin Vyrd) olabilir. VyrdMC model denetleyiciyi bir test senaryosunun koşutzamanlı bir çevrede oluşturacağı tüm yürütmeleri oluşturmak için kullanır. Birçok model denetleyici tarafından yararlanılan kısmi-sıra indirgeme (*partial-order reduction*) yöntemleri [9] doğruluk kriteri açısından nicel olarak eş olan yürütmelerden sadece birini inceleyerek, toplam incelenecek yürütme sayısını büyük ölçüde azaltmaktadır. Bölüm 3.5'de anlatıldığı gibi kısmi-sıra indirgeme yöntemleri bileşenin iç veri yapısı ve arıtma kriteri göz önünde bulundurularak geliştirilebilmektedir.

3.3 Vyrd: Arıtma Doğrulama Aracı

Vyrd, koşutzamanlı çalışan bileşenleri arıtma için denetleyen bir çalışma zamanı doğrulama aracıdır [4]. Vyrd, bileşenin testi sırasında günlüğe kaydedilen yürütme izini eşzamanlı olarak okuyarak atomik belirtime uygunluğunu denetler.
Vyrd, test programına paralel ayrı bir süreç içerisinde günlükteki yürütmeyi oluşturan eylemleri sıralı olarak okuyarak çalışır. Test sırasında oluşan eylemler günlükten farklı zamanlarda okunduğu için Vyrd test programının eylem sonlandıktan hemen sonraki durumuna ulaşamaz. Bu nedenle bu olayları bileşenin farklı bir örneği üzerinde yeniden canlandırmak zorundadır. Bu ek gerçekleştirim

örneği, yürütme izi boyunca meydana gelen ve günlükten okunan eylemlerin örnek üzerine tekrar uygulanmasıyla güncellenir. Vyrd, buna ek olarak, belirtime ait bir örneği de aynı zamanda çalıştırır. Belirtime ait örnek, yürütme izindeki metot çağrılarının bilgisi kullanılarak belirli noktalarda aynı metotların atomik versiyonlarının bu örnek üzerinde çalıştırılmasıyla sürülür.

Arıtma, yürütme izi boyunca teslim (*commit*) noktaları denilen belirli noktalarda denetlenir. Teslim noktaları programcı tarafından metodun kaynak kodu üzerinde bazı kod parçalarının teslim eylemi olarak işaretlenmesiyle belirlenir. Her metot yürütmesi için işaretlenen satırlardan yalnızca biri "belirli koşullar altında teslim eylemi olarak çalışır ve teslim eyleminin çalıştırılması sonucu yürütme teslim noktasına ulaşmış olur. Teslim noktalarının seçimi konusu [3] ve [4] tarafından ayrıntılı olarak incelenmiştir. Çalışan her bir metot için çağrım ve dönüş noktaları arasında "yalnızca" bir adet teslim noktası test programının çalışması sırasında belirlenip günlüğe kaydedilir. Vyrd günlük boyunca ulaştığı her teslim noktasında girdi/çıktı-arıtma ve görünüm-arıtma denetlemesi yapar.

Denetleme için öncelikle teslim noktasının ait olduğu metodun atomik versiyonu aynı argümanlar ile belirtim örneği üzerinde çalıştırılır ve metodun dönüş değeri ile ortaya çıkan belirtim durum bilgisi ele alınır. Girdi/çıktı-arıtma gerçekleştirim ve belirtime yapılan metot çağrılarından dönen değerlerini karşılaştırır. Bu durum hataların yakalanabilmesi için test sırasında bileşenin veri yapısı hakkında bilgi döndüren metotların sıklıkla kullanılması gerekmektedir. Görünüm-arıtma, yürütme izi boyunca teslim noktalarında gerçekleştirim ve belirtime ait durumların ortak bir soyut durum uzayındaki karşılıklarını belirli bir soyutlama fonksiyonu (*abstraction function*) kullanılarak elde eder. Görünüm-arıtma kriteri, elde edilen soyut durumlar arasında yürütme izi boyunca bir denklik ilişkisinin korunmasını gerekli kılmaktadır. Soyutlama fonksiyonu programcı tarafından bileşenin içsel veri yapısını sorgulayarak, bileşenin yalnızca genel metotları aracılığıyla gözlenebilecek bilgiyi içeren daha basit bir veri yapısı döndürecek bir metot olarak yazılır. İlgili yöntem soyut durumu temsil eden bu veri yapısına ait örneklerin biçimsel olarak karşılaştırılabilir olmasını şart koşar. Dönüş değerleri ya da soyut durumlar arasındaki herhangi bir uyumsuzluk bir arıtma hatası olarak tespit edilir ve programcıya ilgili hatayı oluşturan yürütme izi ile birlikte bildirilir.

3.3.1. Deneyimler

Vyrd Boxwood projesi kapsamındaki koşutzamanlı çalışan bileşenlerin ve Java standart kütüphanesinde bulunan birkaç veri yapısını gerçekleştiren sınıfların sınanması amacıyla kullanılmıştır.

Boxwood Projesi: Yapılan çalışmada Microsoft Research tarafından geliştirilen, Boxwood [6] adlı soyut ve dağıtımlı depolama alt yapısına ait bileşenler denetlenmiştir. Vyrd kullanılarak BLinkTree adında koşutzamanlı bir b-link ağaç gerçekleştirimi ve yardımcı bileşenlere arıtma doğrulama uygulanmıştır. BLinkTree düğümleri temsil eden değişkenler byte dizileri olarak Cache adında bir önbellek modülü üzerinden Allocator adındaki soyut bir depolama modülünde saklanır. BLinkTree her biri bir byte dizisi olan [anahtar,değer] ikililerinin en hızlı biçimde saklanmasını ve istenildiğinde geri getirilmesini sağlayacak şekilde eniyilenmiş ileri düzey koşutzamanlı algoritmaları [7] gerçekleştirir.

Boxwood'un BLinkTree ve Cache bileşenleri arıtma için sınanmış ve Cache'in önceki bir versiyonu üzerinde yapılan arıtma sınama sonucunda daha önce fark edilmeyen bir senkronizasyon hatasıyla[2] karşılaşılmıştır. Hata, *Write* ve *Flush* adında iki metodun koşutzamanlı olarak çalışması sonucu belirli bir iş parçacığı zamanlaması sonucunda ortaya çıkmaktadır. Sonuçta Cache'e *Write* metodu ile yazılmak istenen veri *Flush* metodunun bazı işlemlerinin araya girmesiyle diske yanlış olarak kaydedilmekte ve veri bozulması yaşanmaktadır. Bu hatayı test bazlı ya da girdi/çıktı arıtma ile tespit etmek ya da hata bulunsa bile sorunu teşhis etmek çok zorken, görünüm-arıtma hatayı ve neden olan yürütme parçasını hatalı durum oluştuğu anda programcıya bildirmiştir.

Java Standart Kütüphanesi: Java standart kütüphanesi kapsamında StringBuffer ve Vector sınıflarına uygulanan arıtma sınaması sonucunda Vyrd'in daha önceki çalışmalarda tespit edilen hatalar Vyrd tarafından da tespit edilmiştir. StringBuffer'daki hata için [1]'e, Vector'deki hata için [8]'e bakınız. StringBuffer'da bulunan hata, *append* metoduna argüman olarak girilen bir StringBuffer nesnesinin metodu çalıştıran iş parçacığından farklı bir iş parçacığı tarafından aynı anda güncellenmesidir. Bu durum *append* metodu sonucu ortaya çıkan StringBuffer'ın içeriğinin atomik işletim sonucu beklenen içerikten farklı olmasıdır. Vector'de yakalanan hata ise *lastIndexOf* metodunun içsel veri yapısına ait bazı değişkenleri uygun şekilde korumaması nedeniyle *addElement* metoduyla koşutzamanlı çalışması sonucu yanlış değer döndürmesidir.

3.4 Model Denetleyicinin Çalışma Ortamından Yararlanılması

Testin model denetleyici tarafından çalıştırılması sırasında yürütmenin izlenerek atomik çalışan işlemlerin birer eylem olarak günlüğe kaydedilmesi gerekmektedir. Bu amaçla (1) test sırasında atomik eylem olarak yorumlanacak kod parçalarının önceden belirlenmesi ve (2) test senaryosunun çalışması sırasında bu eylemlerin çalışmasının izlemesi ve günlüğe ilgili bilginin kaydedilmesi gerekmektedir.

VyrdMC çerçevesi, kaynak kodun donatılması (*instrumentation*) olarak programcıya ek yük getirecek bu görevlerin, model denetleyicinin çalıştırma ortamından faydalanılarak otomatik olarak kotarılmasını sağlar. Model denetleyici tarafından programların kontrollü olarak çalıştırılması amacıyla kullanılan bu ortam, eylem olarak kabul edilebilecek kod parçalarını otomatik olarak tanımlar ve bu ön-tanımlı kod parçalarının çalışmasını izleyerek gerekli bilgiyi günlüğe kaydeder. Programcı VyrdMC'den, model denetleyicinin atomik olarak yürüttüğü temel işlemler dışında, doğrulanan bileşen tarafından kullanılan diğer bazı yardımcı bileşenleri atomik olarak kabul etmesini isteyebilir. Böylece bu bileşenlere yapılan metot çağrıları model denetleyici tarafından herhangi bir kontrol mekanizması uygulanmadan atomik olarak çalıştırılır. Ayrıca model denetleyici, Bölüm 3.3'de anlatıldığı gibi günlükten okunan eylemlerin gerçekleştirim ve belirtimin Vyrd tarafından kullanılan ayrı iki örneğini üzerinde uygulanmasını da çalışma ortamı aracılığıyla otomatik olarak gerçekleştirilebilmektedir.

3.5 İleri Doğrulama Yöntemleri

Bu bölümde karmaşık veri yapıları içeren koşutzamanlı çok sayıda bileşenin katmanlı bir mimaride bir araya getirildiği endüstriyel ölçekte yazılımların doğrulama sürecini daha verimli hale getirecek teknikler tanıtılacaktır.

[2] İlgili hata Cache bileşeninin son versiyonunda ortadan kaldırılmıştır, ancak yapılan düzeltme önceki versiyondaki hata fark edilmeden gerçekleştirilmiştir.

Katmanlı bileşenlerin doğrulanması: Karmaşık yazılımların doğrulama çalışmalarında önemli bir hedef de katmanlı mimariyle bir araya getirilen bileşenlerin verimli olarak denetlenmesidir. Bu tür yazılımlarda her bileşen bir alt katmandaki bileşenleri soyut bir veri yapısı olarak ya da alt düzey işlemler gerçekleştiren atomik olduklarını varsayarak kullanır. Bunu yaparken alt katmandaki bileşenlerin arıtma kriterine uygun olarak atomik çalışacak şekilde gerçekleştirildiğini varsayar. Ayrıca bir üst katmandaki kendisini kullanan bileşenlere de atomik bir arayüz sağlamalıdır. Programcı klasik varsay-garantile yaklaşımında her bir bileşeni çevresinden bağımsız olarak denetler: Her bir bileşeni denetlemek için ayrı bir test ve doğrulama süreci, gereğinden fazla doğrulama zamanı ve kaynağı harcanmasına sebep olacaktır.
VyrdMC, katmanlı bir mimaride birleştirilmiş bir bileşen kümesinin tümünü aynı çalışması sırasında doğrulamaya izin vermektedir. Bu durumda tüm katmanların koşutzamanlı olarak çalışması sağlanır. Ancak her bir bileşen için ayrı denetleme kaynağı (örneğin her bileşen için bir gerçekleştirim ve bir belirtim) ayrılır. test birimi ve model denetleyici ortak olarak kullanıldığı için ve tüm bileşenler için tek bir test senaryosu ve ortak bir yürütme izi kullanıldığı için toplam kaynaktan tasarruf söz konusu olmaktadır. Her bir bileşenin denetlemesi aynı yürütme izi kullanılarak kendisine ait gerçekleştirim ve belirtim üzerinde arıtma kriterleri göz önünde bulundurularak yapılır. Tüm bileşenlere ait yürütme izi parçalarının doğruluğunun gösterilmesi, tüm yürütmenin doğruluğunu ispatlar.

Kısmi-sıra indirgeme tekniklerinin geliştirilmesi: Kısmi-sıra indirgeme metotları [9,10] eylemler arasındaki bağımlılık ilişkilerini nicel olarak farklı yürütmeler oluşturmak amacıyla kullanırlar. Doğruluk kriteri zamansal mantık (*temporal logic*) formülleri ya da kilitlenme (*deadlock*) olan yaklaşımlar bu metotları sıkı veri bağımlılığı kuralları ile birlikte kullanırlar. Bu durumda oluşabilecek farklı yürütmelerin sayısı doğrulama süresini önemli ölçüde artırmaktadır. VyrdMC tüm bu yürütmelerin arıtma için ilginç olan bir altkümesini inceleyerek senkronizasyon hatalarının daha kısa sürede tespit edilmesine imkan tanır.
Öncelikle bileşenlerin içsel veri yapıları arıtma kriterinde etkisine bağlı olarak farklı parçalara ayrılır. Örneğin veriyi yapraklarda saklayan bir ağaç veri yapısı (1) yaprak düğümleri ve (2) dizin yapısını oluşturan düğümleri temsil eden değişkenler olarak iki kümeye ayrılabilir. Farklı değişken kümeleri üzerinde uygulanan işlemler arasındaki bağımlılık ilişkilerinin kısmi-sıra indirgeme algoritması tarafından farklı şekilde değerlendirilmesi sağlanarak bazı bağımlılıklar ihmal edilebilir. Örneğin model denetleyici sadece doğruluk kriteri için ilginç olabilecek değişken kümesi üzerine yapılan eylemler arasındaki bağımlılık ilişkileri değerlendirerek incelenecek toplam yürütme sayısını önemli ölçüde azaltabilir. Sonuçta sadece ilgilenilen değişken kümesi üzerinde çalışan eylemlerin farklı permütasyonları denetlenir. Bu yöntem özellikle karmaşık veri yapıları ve bu yapılar üzerinde işlem yapan kapsamlı algoritmalar içeren bileşenler için, istenilen değişkenlere ulaşan kod parçalarının koşutzamanlı işleyişleri üzerine odaklanılması gerektiren denetlemeler için büyük önem taşımaktadır.

4. Doğrulamanın Geliştirme Sürecine Entegrasyonu

VyrdMC hem yazılımı oluşturan her bir bileşeni arıtma kriterlerine göre denetlerken, hem de yazılımın katmanları arasındaki etkileşimin birleşik olarak doğrulanmasını (*compositional verification*) sağlanmaktadır. Çerçevenin test birimi az sayıda metodun koşutzaman karakteristiğinin kısa süreler içerisinde denetlenebilmesine imkan sağlar. Bu durumda programcı bileşene eklediği senkronizasyon mekanizmalarının doğruluğunu VyrdMC'yi çalıştırarak

sınayabilir ve bariz hataları geliştirme sürecinin ilk aşamalarında tespit edip düzeltir. Koşutzamanlı çalışacak metot sayısı artırılarak daha karmaşık durumlarda oluşacak hatalar bulunabilir.
Denetleme süreci programcı için en az düzeyde katkıda bulunacak şekilde gerçekleşmektedir. Programcının sorumlulukları doğrulanacak her bileşen için aşağıdaki gibi özetlenebilir:

1. Bileşenin durumunu girdi olarak alıp soyut durumu seçip çıkartan bir soyutlama fonksiyonu yazmak. Bu fonksiyon bileşenin bir metodu olarak gerçekleştirimle aynı programlama dilinde yazılır. Soyutlama fonksiyonunun görevi bileşenin durumunu tarayarak soyut durumu temsil edecek daha basit bir veri yapısı örneği oluşturmak ve bunu arıtma denetleyiciye döndürmektir.
2. Her metot için teslim noktalarının belirlenmesi ve bunların kaynak kod üzerinde işaretlenmesi. Teslim noktaları gerçekleştirimin yürütmesine karşılık gelen belirtime ait yürütmenin oluşturulması için kullanılacağından bileşenin doğruluğunu göstermek açısından önem taşımaktadır.

Programcı sınanacak bileşenin içsel veri yapısına karar verdikten sonra bileşenin soyutlama fonksiyonunu yazar ve bu fonksiyon veri yapısında önemli bir değişiklik yapılmadıkça değişmez. Ancak teslim noktaları ilgili metotların gerçekleştirimindeki değişikliklere bağlı olarak birçok kez değişecektir.
VyrdMC'yi programcıya yüklediği yukarıda belirtilen sorumluluklar her bileşen için ek yük getiriyor gibi gözükebilir. Ancak gerek soyutlama fonksiyonu gerekse commit noktaları, programcının gerçekleştirimin özellikle koşutzamanlı bir ortamda doğruluğundan emin olmak için üzerinde düşünmek zorunda olduğu gereksinimlerdir. Programcı bu gereksinimleri VyrdMC tarafından kullanılmak üzere biçimsel olarak ifade eder. Sınama sırasında soyutlama fonksiyonu ya da commit noktalarının yanlış ya da yetersiz ifadesi sonucu ortaya çıkan hatalar programcının bileşenin tasarımını geliştirmesi açısından önem taşıyacaktır.

5. Sonuç

Bu bildiride koşutzamanlı bileşenler için yapısal bir test birimi, model denetleyici ve Vyrd arıtma denetleme aracını içeren bir doğrulama çerçevesi, VyrdMC, sunulmuştur. Test birimi doğrulama işlemi için gereken kaynak ihtiyacını en aza indirecek ve hatalı kısımlara odaklı test senaryoları üretir. Üretilen test senaryoları model denetleyici tarafından kapsamayı artıracak farklı yürütmeler oluşturacak şekilde çalıştırılır. Model denetleyicinin çalışması sırasında Vyrd oluşturulan yürütme izlerini arıtma kriterlerine göre denetler.
VyrdMC katmanlı bir mimari sergileyen bileşenleri de aynı çalışmada doğrulayabilmekte ve kısmi-sıralı indirgeme tekniklerini geliştirerek karmaşık veri yapıları içeren bileşenler için yürütme uzayını makul büyüklüklere indirgeyebilmektedir. Önerilen çerçeve koşutzamanlı bileşenlerle kurulan yazılımları bileşen bazında otomatik olarak denetleyerek programın tümünün doğruluğu hakkında fikir yürütmek açısından büyük önem taşımaktadır. Çerçevenin, koşutzaman hatalarının önceden tespit edilip düzeltilmesi amacıyla yazılım geliştirme sürecine entegre edilmesi tavsiye edilmektedir.

Kaynakça

[1]. C. Flanagan ve S. N. Freund. Atomizer: A dynamic atomicity checker for multithreaded programs. Proc. 31st ACM Symposium on Principles of Programming Languages, sayfa: 256–267, 2004.

[2]. C. Flanagan, S. Freund, ve S. Qadeer. Exploiting purity for atomicity. Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2004). ACM Press, 2004.

[3]. S. Tasiran ve S. Qadeer. Runtime refinement verification of concurrent data structures. Proc. Runtime Verification '04 (ETAPS '04). Electronic Notes in Theoretical Computer Science. Elsevier, 2004.

[4]. Tayfun Elmas, Serdar Tasiran, ve Shaz Qadeer. Vyrd: VerifYing Concurrent Programs by Runtime Refinement-Violation Detection. ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation (PLDI), Chicago, Illinois, ABD, Haziran 12-15, 2005.

[5]. Tayfun Elmas, Serdar Tasiran. VyrdMC: Driving Runtime Refinement Checking with Model Checkers, Fifth Workshop on Runtime Verification (RV'05). The University of Edinburgh, Scotland, UK. Temmuz 12, 2005.

[6]. J. MacCormick, N. Murphy, M. Najork, C. A. Thekkath, ve L. Zhou. Boxwood: Abstractions as the foundation for storage infrastructure. Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI 2004), San Francisco, CA, ABD, Aralık 2004, sayfa: 105-120.

[7]. Y. Sagiv. Concurrent operations on b-trees with overtaking. Journal of Computer and System Sciences, 3(2), Ekim 1986.

[8]. C. Flanagan ve S. Qadeer. A type and effect system for atomicity. In Proceedings of the ACM SIGPLAN 2003 Conf. on Programming Language Design and Implementation, PLDI ’03.

[9]. P.Godefroid. Partial Order methods for the Verification of Concurrent Systems-An approach to the State-Explosion Problem. Sayı: 1032 of Lecture Notes in Computer Science. Springer-Verlag, Ocak 1996.

[10]. Cormac Flanagan, Patrice Godefroid. Dynamic partial-order reduction for model checking software. POPL '05: Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, Long Beach, California, ABD. Sayfa: 110--121, 2005.

[11]. Guillaume Brat, Klaus Havelund, Seung-Joon Park, ve Willem Visser. Model checking programs. IEEE International Conference on Automated Software Engineering (ASE), Eylül 2000.

[12]. Eitan Farchi, Yarden Nir, Shmuel Ur. Concurrent Bug Patterns and How to Test Them. 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), 22-26 Nisan 2003, Nice, Fransa, sayfa: 286.