Güvenlik açısından, Yazılım Tasarımı ve Yazılım Ürün Doğrulaması

Ufuk Çağlayan, Albert Özkohen
caglayan@boun.edu.tr, albert.ozkohen@boun.edu.tr

Boğaziçi Üniversitesi
Bilgisayar Mühendisliği Bölümü

28.12.2012

Özet

Mevcut bir yazılım kaynak kodu ile onunla ilgili tasarım ürünü arasındaki denkliğin tanımlanması için birçok çalışma yapılmaktadır. Tasarım ve uygulama yazılım yaşam döngüsünün önemli iki aşamasıdır. Verilmiş bir yazılım kaynak kodu ile UML ile oluşturulmuş bir tasarımı, SPIN Model Denetleyicisinin dili olan PROMELA diline çevirerek, bu ikisi arasındaki denkliği, iki sonlu durum makinasının denkliği ile gösterebileceğimizi öneriyoruz

1 Giriş

Yazılım mühendisliğinde, yaşam döngüsünün çeşitli evreleri vardır. Yazılımın uygulanması sırasında, yazılım tasarımı girdi olarak kullanılır. Bu süreç sonunda C, Java ya da başka bir dilde yazılmış yazılım kaynak kodu ortaya çıkar. Problemimiz, tasarımı verilen bir yazılım kaynak kodunun bu tasarımda belirtilen tüm özellikleri yerine getirip getirmediğini ispat eden bir mekanizma ortaya koymaktır.

Bu problem, yazılım mühendisliği biliminin ortaya çıkışından itibaren vardır. Üstelik mevcut yazılım sistemleri boyut ve işlevsellikleri arttıkça, bu sorun daha da büyümekte ve çözümü önem kazanmaktadır.

Büyük ve karmaşık yazılım sistemleri, küçük sistemlere göre hem sayıca daha çok hem de belirsizlik seviyesi daha yüksek hatalar içerirler.

Yazılım geliştirme evresi, tasarımdan sonraki evredir. Yazılım geliştiren birçok kurum, yazılım kaynak kodlarında değişiklikler yaparken, bu değişiklikleri ilgili tasarımlara yansıtmamaktadırlar. Böylelikle, yazılım geliştirme esnasında birçok projede tasarım ve kaynak kodları kendi aralarında farklı ve tutarsız hale gelebilmektedir.

Ayrıca, bu değişikler tasarıma yansıtılsa bile, bu yansımalar sonucunda tasarım ile kaynak kodunun denkliğini ispat edemez. Özellikle güvenlik alanında Yazılım Doğrulaması hayat önem taşımaktadır

Otomatik kaynak kodu üreten birçok ticari araç bulunmaktadır. Bu işlem basit tasarımlarda olabilmektedir ancak henüz genelleştirilmiş değildir. Zira hala endüstride yazılım geliştirme uzmanları bulunmakta ve faaliyet göstermektedirler.

Tam tersi durumda henüz otomatikleştirilmiş değildir. Yazılım kodu verildiğinde otomatik tasarım üreten bir sistem henüz mevcut değildir. "Tersine Mühendislik" adı altında bazı girişimler vardır ancak bu her durumda geçerli bir evrensel tasarım üretme işini henüz yapamamaktadır.

Bu çalışmadaki ana katkının, verilmiş bir yazılım kaynak kodunun, verilmiş bir tasarımdaki tüm özellikleri ve gereksinimleri sağladığımım doğrulanması olması amaçlanmaktadır. Halen endüstride mevcut yazlım kodları ve ilişkili tasarımlar kendi aralarında tutarlı değildir. Çalışmamız bunların denk olup olmadığının belirleyecek ve mümkünse kısmı denkliğin nerelerde olduğu / olmadığının sağlayacak bir mekanizma oluşturmayı hedeflemektedir.

2 Yazılım Doğrulaması hakkında literatür taraması

Yazılım geliştirme çalışmaların ilk zamanlarından itibaren, yazılım tasarım yapılarının ortak bir biçim ile yapılması hakkında araştırmalar yapılmıştır. Booch, Rumbaugh ve Jacobson Birleştirilmiş Modelleme Dilini (UML) bu amaçla önermişlerdir. [Boo96]. Kullanım senaryoları (Use Case) ana aktörleri ve rollerini belirlemektedir. Sınıf diyagramları, verinin yapısını tanımlamaktadır. Zaman akış diyagramı sistemlerin zamandaki davranışlarını, faaliyet diyagramı çalışma olasılıklarını, paket diyagramı uygulamanın dağıtılmasını tanımlayan bileşenlerdir. Başka alternatifler olsa da UML yazılım tasarımı için endüstri olmuştur.

UML biçimsel değildi. Bu yüzden UML için biçimsel gösterimlerin oluşturulması adına çeşitli çalışmalar yapıldı. OCL (nesne kısıt dili) bunlardan bir tanesidir. Richters OCL dilinin tanımını ve biçimselliğini [Ric98] de yapmaktadır. Bu çalışma UML diline biçimsel bir anlam getirmektedir.

Mezei ve arkadaşları [Mez07] de OCLASM adında yeni biçimsellik tanımladılar. Bu çalışmada, OCL dili Soyut Durumsal Makineler bazında tanımlanarak, OCL kısıtlarının dinamik davranışları modellendi. Böylece OCL dili sadece statik değil dinamik gösterimler için de kullanılabildi.

Güvenlik alanında da tasarım mekanizmaları oluşturmak için birçok çalışmalar yapılmıştır. Jürjens UML dilinin bir uzantısı olarak UMLsec dilini [Jür02] de tanımlamıştır. Jürjens bu çalışmada UML in standart uzantı mekanizmalarını kullanmış ve bunları profil olarak tanımlamıştır. Bu profillerle sistemlerin güvenlik özelliklerin daha iyi ve bütünsel olarak tanımlayabilen özelleştirilmiş diyagramlar oluşturmuştur. UMLSec Genel olarak kabul edilen bir biçimselliktir, ancak UML de yazılmış tüm

tasarımların biçimsel UMLsec formuna dönüştürülmesi gerekmektedir ki bu süreç hem pratik değildir hem de tam olarak çözülememiştir.

Buchholtz ve arkadaşları, [Buc08] çalışmasında For-LySa adında yeni bir yöntem önermektedirler. Bu çalışmada Buchholtz kimlik doğrulama için özel tasarımları modellemişlerdir. Lysa adında bir Süreç Analiz metodu uygulamaktadırlar ve bu metodu UML özelleştirilmiş profillerinde kullanmaktadırlar

[Pav07] çalışmasında Pavlich ve arkadaşları, erişim denetimi alanında özelleştirilmiş özel diyagramlar önermektedirler. Bu diyagramlar, rol-bazlı zorunlu ya da seçenekli erişim denetimi mekanizmaları tanımlayarak, UML in güvenlik yeteneklerini arttırmışlardır, Ancak, tüm tasarımlar UML in bu özelleştirilmiş versiyonuna hazırlanmış değildir ve yine bunun için özel bir otomatik tercüme mekanizması mevcut değildir

Model denetimi Clarke tarafından [Cla97] çalışmasında tanımlanmıştır. Sonlu durum tabanlı tepki veren sistemlerin doğrulanması için bir tekniktir. Özellikler zamansak mantık formülleri olarak ifade edilir ve bu özellikler üretilen durumların durum geçiş grafikleri kullanılarak denetlenir. Durum sayılarının olası büyüklüklerine karşı, durum sayısı indirgeme teknikleri de önerilmiştir

SPIN Holzmann tarafından [Hol97] çalışmasında yazılım sistemlerinde Model Denetimi aracı olarak önerildi. SPIN in kullandığı dil PROMELA dilidir (Proses Mete Dili). Bu dil, PROMELA da tanımlanan bir sistemin tasarım özelliklerini karşılayıp karşılamadığını denetler.
Jürjens ve Shabalin [Jür04] çalışmasında, bir UMLsec modelini, gereksinimlerine karşı biçimsel olarak doğrulamayı başardılar. UMLsec dilinden PROMELA eşdeğerine çevirdiler. Onların yaklaşımlarını UML tasarımlarınızı PROMELA koduna çevirirken kullanacağız.

Corbett ve arkadaşları [Cor02] çalışmasında altyapıda Java kaynak kodunu PROMELA ’ya çevirebilen, Bandera isimli bir yazılım altyapısı geliştirdiler. Bu çalışmada bir yazılım soyutlaması önerdiler ve özellikleri ifade eden bir dil geliştirdiler. Bu dil yazılım üretecine benzer bir bileşene girdi göndererek çevirme işlemini gerçekleştirdi. Aynı yaklaşım C kodunu PROMELA ya çevirirken kullanılabilir
.
Havelund Java PathFinder [Hav00] isimli aracını önerdi. Bu araç, NASA 'nın kritik görevdeki sistemleri için Java kodunu PROMELA ya çevirdi. Her diyagram tipi özelleştirilmiş bir algoritma ile ele alındı.

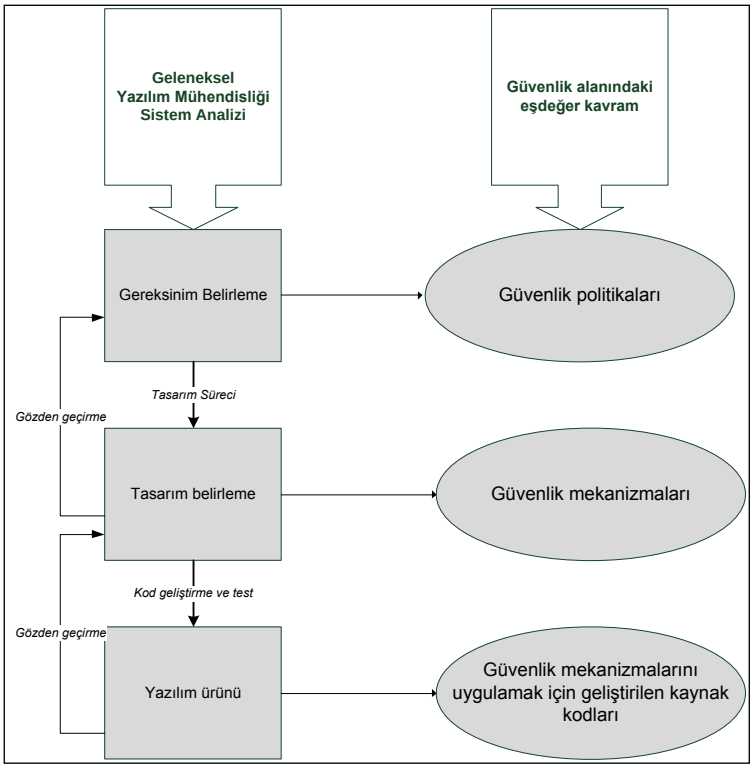
Kaliappan [Kal08] çalışmasında, iletişim protokollerinin doğrulanması için farklı bir yöntem önerdi. Her UML durum diyagramı PROMELA ya çevirdi ve onunla ilgili zaman akış diyagramını doğrusal zamansal mantığa çevirdi (LTL) . Bunları SPIN sistemine girdi olarak vererek sistemde bir tutarsızlık olup olmadığını model denetimi yöntemiyle kontrol etti. Bu yöntem iyi gözükse de her durum diyagramının yanında zaman akış diyagramı olmayabilir ve sınıf diyagramları da bu çalışma dışında bırakılmıştır.

Sutton [Sut05] çalışmasında C++ bileşenleri ve komutları ile onlara karşı gelen UML bileşenleri (örneğin ilişkiler, çokluk, toplama) için kurallar önerdi.

3 Yazılım doğrulaması seçenekli model önerileri

Verilen bir tasarımın verilen bir kaynak kodu ile denk olup olmadığını ölçecek iki farklı model öneriyoruz. Önce Yazılım Mühendisliği yaşam çevrimindeki çeşitli faaliyet alanlarını özetleyelim. **(Bkz** Error! Reference source not found**.)**

Yazılım mühendisliğinde, yaşam döngüsünün gereksinim analizi, tasarım, yazılım geliştirme gibi çeşitli evreleri vardır. Yazılımın uygulanması sırasında, yazılım tasarımının sonuçları yazılım geliştirme işleminde girdi olarak kullanılmaktadır. Bu sürecin sonunda C, Java ya da başka bir dilde yazılmış yazılım kaynak kodu ortaya çıkacaktır. Çözümünü bulmaya çalıştığımız problem, tasarımı önceden verilmiş bir yazılım kaynak kodunun bu tasarımda belirtilen, ne eksik ne fazla, tüm özellikleri birebir yerine getirip getirmediğini bulan ve ispat eden bir mekanizma ortaya koymak ve geliştirmektir.



Şekil 1 Yazılım geliştirme yaşam çevrimi evreleri

Yazılım kodu ile sağlanan özelliklerin tasarıma göre daha fazla ya da eksik olması denklikte sorun olacaktır. Ek işlevler, gerektiği gibi denetlenmediğinde güvenlik ihlalleri ya da tutarsız durumlar oluşturabilir, Sağlanmaması işlevler ise tasarıma uygunsuzluk olacaktır. Dolayısıyla, tasarım ile kaynak kodu arasında birebir uygunluk önemlidir. Bu çalışmada, kısmi bir denklik olup olmadığının tespiti de hedeflenmektedir. Bu durumda uyumsuzluk yaratan parçaların neler olduğunun tespiti de gerekecektir.

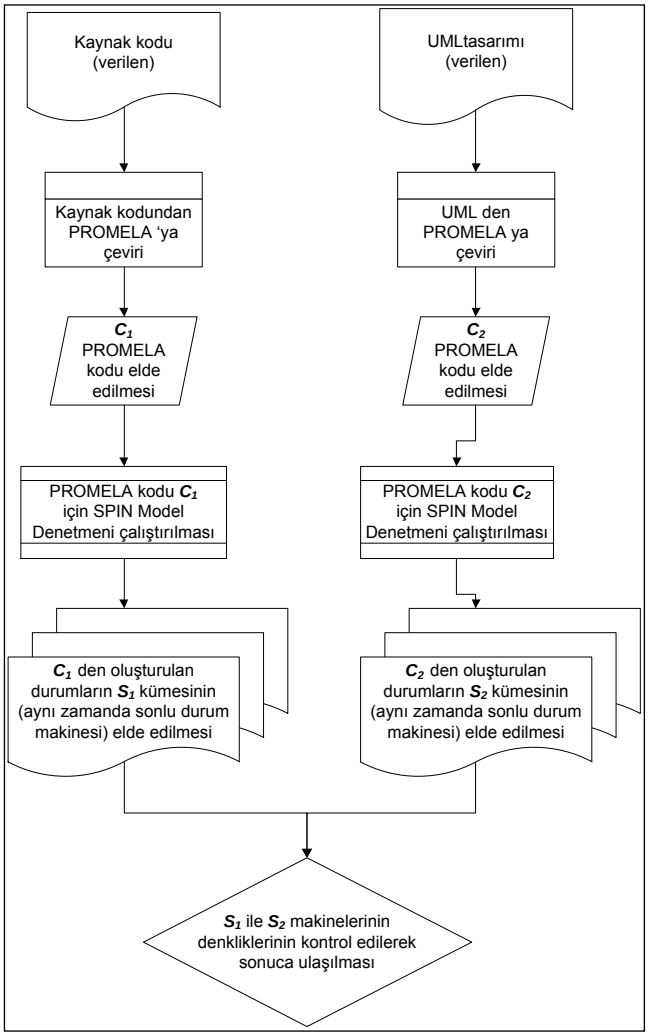
Kaynak kodunu tasarımına karşı doğrulayan ve biçimsel olarak tam ve eksiksiz bir sistemin genel amaçlı olarak yaratılması sırasında birçok yan etki ortaya çıkacaktır. Tüm alanlar için bu problem çözmeye çalışmak çok karmaşık olacaktır. Bu yüzden, ekonomik ve teknolojik açılardan gittikçe daha önemli olan güvenlik alanında bu işlemi yapmayı hedefledik. Daha da ayrıntıya indiğimizde sistemi güvenlikte erişim denetimi alt alanında özelleştireceğiz

3.1 Seçenek-1

Bir kaynak kodu ve UML tasarımı girdi olarak verilecektir. Amacımız ikisini de PROMELA ya çevirmektir. PROMELA, SPIN model denetimi sisteminin dilidir. Literatür tasarımında bu işlemi yapabilen birçok çalışma mevcuttur.

Bu çevirilerden sonra iki tane PROMELA kodumuz olacaktır. Bu kodları SPIN de çalıştıracağız. SPIN bu kodların ulaşabileceği tüm durumları içeren durum aşamaları oluşturacaktır. Bu durum alanları aslında sonlu durum makineleridir (Finite State Machine). Bunlar elimizde olunca bu iki makinenin eşdeğer olup olmadığını bulacağız.

Böylece problemimiz aslında iki sonlu durum makinesinin eşdeğer olup olmamasının bulunmasına indirgenmiş olacaktır.



Şekil 2 - Önerdiğimiz yöntem: 1. seçenek

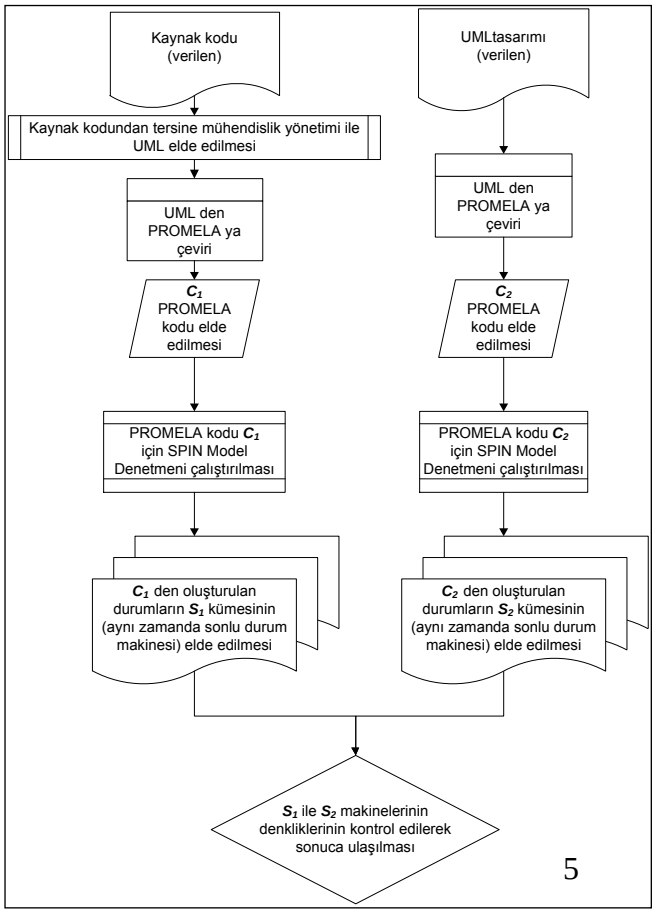
Girdi olarak, güvenlik alanında bir kaynak kodu ve bir UML tasarımı Kabul eden bir yazılım altyapısı oluşturarak, bu iki girdinin oluşturacağı sonlu durum makinelerinin denk olup olmadığının kontrolünü yapan bir çözümünü bulacağız

Bu seçenek **Error! Reference source not found.** de şematik olarak anlatılmıştır.

3.2 Seçenek-2

Bu seçenekte girdimiz yine kaynak kodu ve UML tasarımı olacaktır. Fakat bu sefer kaynak kodunu tersine mühendislik yöntemleriyle UML tasarımına çevirmeye çalışacağız. Bu çeviri için literatürde örnekler mevcuttur.

Daha sonra, bu iki tasarımı (ilk verilen ile tersine mühendislik sonucu elde edilen) SPIN model denetmeninin dili olan PROMELA diline



Şekil 3 - Önerdiğimiz yöntem: 2. seçenek

çevireceğiz. Bu çevirinin literatürde birçok seçeneği bulunmaktadır.

Bu çevirilerden sonra durum seçenek-1 ile aynı yere gelmiş olacaktır ve sonrası aynı seçenek-1 de olduğu gibi ilerleyecektir.

Bu seçenek **Error! Reference source not found.** de şematik olarak açıklanmıştır.

4 Sonuç ve gelecek çalışmalar

Amacımız, güvenlik alanında bir model geliştirmektir. Bu modelde iki girdi olacaktır: yazılım kaynak kodu ve UML tasarımı. Modelimiz sonuç olarak tasarımdaki özelliklerin yazılım koduna birebir eksiksiz ve tam olarak yansıtılıp yansıtılmadığını bulmaktır. Başka bir deyişle eksik ya da fazla özellik olmadığını ispat etmektir

Bu amaçla, iki seçenek önerdik. Literatür taramasında, kaynak kodundan PROMELA yay a da UML den PROMELA ya çeviri yapan sistemler mevcuttur. Ayrıca tersine mühendislik yapan sistemler de mevcuttur. Bunlardan performansını uygun gördüklerimiz sistemimizin parçası olacaktır.

Sonraki çalışma iki sonlu durum makinasının eşdeğer olduğunun nasıl gösterildiğinin üzerinde çalışması ve / veya literatür taraması olacaktır. Temel olarak iki sonlu durum makinasına verilen herhangi aynı girdi, her zaman aynı çıktıyı üretiyorsa ve bu durum biçimsel olarak ispat edilebiliyorsa o zaman bu iki sonlu durum makinası eşdeğerdir.

Bundan sonra kaynak kodu örnekleri ve ilgili tasarımlar verilerek bu modelin nasıl çalıştığı örneklerle gösterilecektir. Bundan yola çıkarak modelimizi her değişik programlama bileşeni için (döngüler, seçimleri komutlar, sınıf tanımları) genelleştireceğiz. Ayrıca kaynak kodunun ne kadar tasarımın ne kadarını doğruladığını gösteren bir metrik de geliştirme hedeflerimiz arsandadır

Sonuçta, yazılım doğrulama alanında araştırmacıları bu modeli diğer alanlara uyarlayarak yazılım kodu ile tasarımların uyumlu / uyumsuz olduğu noktaları tesit eden sistemler gerçekleştirebilecektir.

Referanslar

Buc08: Buchholtz M, Montangero C, Perrone L, Semrini S, And For-LySa: UML for Authentication Analysis, Lecture Notes in Computer Science, Volume 3267, pp 93-106, Springer, 2005.

Boo96: Booch G, Rumbaugh J, Jacobson I, The Unified Modeling Language for Object Oriented Development, Documentation Set, and The Rational Software Corporation 1996.

Cla97: Clarke E, Grumberg O, Long D, Model Checking, Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, Volume 1346, pp 54-56, Springer 1997.

Cor02: Corbett J, Dwyer M, Hatchclff J, Laubach S, Pasareanu C, Robby, Zheng H, Bandera: Extracting Finite-State Models from Java Source code, Proceedings of the International Conference on Software Engineering, pp 439-448, 2000

Hav00: Havelund K, Prerssburger T, Model checking JAVA programs using JAVA PathFinder, Int J STTT, pp 366-381, Springer Verlag, 2000

Hol97: G.J. Holzmann The model checker SPIN, IEEE Trans. on Software Engineering, Vol. 23, No. 5, pp. 279-295, May 1997

Jür02: Jürjens J, Lecture Notes in Computer Science, Volume 2460, pp 412-425, Springer, 2002.

Jür04: Jürjens J, Shabalin P, Automated Verification of UMLsec Models for Security Requirements, Lecture Notes in Computer Science, Volume 3273, pp 365-379, Springer 2004.

Kal08: Kaliappan P S, Koenig H, Designing and Verifying Communication Protocols Using Model Driven Architecture and Spin Model Checker, Journal of Software Engineering and Applications, Issue 1, pp13-19, 2008

Mez07: Mezei G, Levendovsky T, Charaf H, Formalizing the Evaluation of OCL Constraints, Acta Polytechnica Hungarica, Vol4 No.1, pp.89, 2007.

Pav07: Pavlich-Mariscal J, Michel L, Demurjian S, Enhancing UML to Model Custom Security Aspects, Aspect-oriented Modeling Workshop Models 2007.

Ric98: Richters M, Gogolla M, On formalizing the UML Object Constraint Language OCL, Proc. Of 17th Intl Conference on Conceptual Modeling, 1998.

Sut05: Sutton A, Maletic J I, Mappings for Accurately Reverse Engineering UML Class Models from C++, Proceedings of the 12th Working Conference on Reverse Engineering, 2005.