Referans Mimariye Uygunluğun Model Tabanlı Analizi İçin Bir Yazılım Aracı

Evren Çilden

Aselsan, PK. 1, 06172 Yenimahalle,
Ankara Türkiye
ecilden@aselsan.com.tr

Özet. Yazılım geliştirme süreci içinde gerçeklenen yazılımın mimarisinde, tasarlanan mimariden çeşitli nedenlerden ötürü sapmalar olabilmektedir. Bu durum kodun anlaşılırlığının azalmasına ve tasarlanan mimaride dikkate alınan kalite faktörlerinin gerçeklenen mimaride artık karşılanamamasına neden olmaktadır. Kod-mimari uyumsuzluğunun mümkün olan en erken aşamada fark edilerek düzeltilebilmesi için kodun manüel yöntemlerle gözden geçirilmesi maliyetli olmaktadır. Yazılımda gerçeklenen mimarinin çeşitli statik analiz yöntemleri kullanılarak çıkartılması ve tasarlanan mimariye uygunluğun araçlar yardımıyla denetlenmesi mümkündür. Bu çalışmada radar kullanıcı arayüzü alanında geliştirmekte olduğumuz yazılımların kullanım görünümü açısından referans mimariye uygunluğunun model tabanlı Yansıma Modeli yöntemi yardımıyla denetlenebilmesi için geliştirilen statik analiz aracı anlatılmıştır. Geliştirilen araç Yansıma Modeli yöntemini kullanan diğer araçlardan farklı olarak, paket seviyesinde değil de bileşen seviyesinde analiz yapılmasına ve hazırlanan mimari modellerde yazılım ürün hattı bileşenlerinin belirtilebilmesine olanak sağlamaktadır.

Anahtar Kelimeler. Yazılım mimarileri, referans mimari, mimari model, mimari erozyon, statik analiz, Yansıma Modeli, yazılım ürün hattı mühendisliği, EMF, GMF

1 Giriş

Yazılım geliştirme yaşam döngüsü boyunca çeşitli nedenlerden ötürü yazılım kaynak kodu ile mimari arasında uyumsuzluklar oluşabilmektedir. Bu durum ilk olarak Perry ve Woolf'un 1992 yılında yayınlanan makalesinde [1] üzerinde çalışılması gereken bir problem olarak dile getirilmiştir. Bu saptamanın üzerinden 14 yıl geçmesine rağmen, Shaw ve Clements'in çalışmasında [2], konu üzerinde biraz ilerleme kaydedilse de henüz çözüme ulaşılamamış olduğu belirtilmiştir.

Yazılım kaynak kodlarında gerçeklenen mimarinin ilk tasarımdan uzaklaşmasına ve kodda mimariye aykırı öğelerin artmasına literatürde mimari erozyon adı verilmiştir [1]. Mimari erozyon oluşmasındaki en büyük etken değişkenlik faktörüdür. Gereksinimler yazılım yaşam döngüsünün hemen her aşamasında değişebilmektedir. Kodlama aşamasında gelen bazı değişikliklerin başlangıçta alınan mimari kararlarla

çelişmesi nedeniyle kodda uygulanan mimari farklılık gösterebilmektedir. Mimari kararların dokümantasyonundaki eksikler de kodda gerçeklenen mimaride farklılaşmaya neden olabilmektedir. Bunun yanı sıra yazılım ekibinde sıklıkla değişiklikler olduğunda mimari kararların ekip üyelerine eksiksiz ve doğru olarak aktarımında sıkıntılar olabilmektedir.

Değişkenlik faktörü dışındaki diğer bir önemli faktör de yoğun zaman baskısıdır. Bir mimari kararın uygulanması başka bir çözüm yoluna göre daha zaman alıcı olduğunda yazılım mimarisi kararları uygulanmadan kısayol çözümler uygulanabilmektedir. Bu duruma yazılım dünyasında *teknik borç* adı verilmektedir [3]. Bu ifadenin kullanılmasının nedeni zaman baskısı oluşturan etkenin ortadan kalkmasından sonra kodda gerekli yeniden yapılandırmalar gerçekleştirilerek mimari kararların uygulanması beklentisidir. Fakat zamanla teknik borç adı altında uygulanan çözümler kodda geçerli çözümler haline gelebilmektedir.

Yazılım kodları ile yazılım mimarisi arasındaki uyumsuzluklar, çeşitli kalite faktörleri gözetilerek tasarlanan mimariden uzaklaşılmış olması nedeniyle, nihai üründe kalite faktörlerinin karşılanamamasına neden olmaktadır. Ayrıca tasarlanan mimariden sapmaların kodun anlaşılırlığının azalması, değişiklik maliyetinin artması gibi etkileri de olmaktadır. Yazılım mimari erozyonunun maliyeti üzerinde bir çalışma US Air Force Software Technology Support Center (STSC) tarafından gerçekleştirilmiştir [4]. Yaklaşık 50.000 satırlık bir yazılımın iki versiyonu değişik ekiplere verilerek yaklaşık 3.000 satırlık bir ekleme gerektiren idame işi yaptırılmıştır. İlk ekibe mimari erozyon etkileri giderilmemiş yazılım kaynak kodları verilmiştir. İkinci ekibe ise mimari erozyon etkileri giderilerek revize edilmiş kaynak kodları teslim edilmiştir. İlk versiyon üzerinde çalışan ekip ikinci ekibe göre işi tamamlamak için iki kat fazla zamana ihtiyaç duymuştur. İlk ekibin ürettiği yazılımda tespit edilen hata sayısı ikinci ekipten sekiz kat daha fazla olmuştur.

Kod-mimari uyumsuzluğunun olumsuz etkilerinden uzak durabilmek için radar kullanıcı arayüzü alanında geliştirmekte olduğumuz yazılımlarda mimariden sapmaların yazılım geliştirme sürecinin erken aşamalarında belirlenmesi ihtiyacı vardır. Bu bildiride bu amaca hizmet etmek üzere geliştirilen ve sadece kullanım bağımlılıkları açısından kod-mimari uyumsuzluklarının belirlenmesini sağlayan araç anlatılmaktadır. Bildirinin 2. bölümünde bu amaçla kullanılabilecek mevcut yöntemler ve araçlar kısaca tanıtılmıştır. 3. bölümde radar kullanıcı arayüzü yazılımları için analiz ihtiyaçlarından bahsedilerek, 4. bölümde bu ihtiyaçlar gözetilerek geliştirilen statik analiz aracı tanıtılmıştır.

2 Mevcut Yöntemler ve Araçlar

Yazılım kaynak kodu ile mimari arasındaki uyumsuzlukların belirlenebilmesi için yazılım kaynak kodlarının incelenerek kullanılan mimari öğelerin, mimari yapının, veri ve kontrol akışının belirlenmesi gerekmektedir. Yazılım kaynak kodu miktarı arttıkça bu belirlemenin kod okuma ve tersine mühendislik yöntemleri ile yapılması

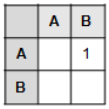
zorlaşmakta, yöntem hataya açık hale gelmektedir. Bu nedenle yazılım kaynak kodlarını ya da yazılımın çalışma zamanındaki davranışlarını analiz ederek mimari özelliklerinin belirlenmesini sağlayan çeşitli yöntemler ve araçlar geliştirilmiştir.

Tablo 1'de kullanım bağımlılıkları açısından mimari uyumluluğun denetlenmesinde kullanılabilecek çeşitli araçlar ve kullandıkları yöntemler verilmiştir [5, 6, 7, 8, 9, 10, 11, 12]. Araçlarda Bağımlılık Yapısı Matrisi (İng. Dependency Structure Matrix - DSM) [13], Bağımlılık Kısıtlama Dilleri (İng. Dependency Constraint Language - DCL) [14] ve Yansıma Modeli (İng. Reflexion Model) [15] yöntemleri kullanılmıştır. Bu bölümde bu üç yöntem kısaca tanıtılacaktır.

Table . Kullanım Bağımlılıkları Açısından Mimari Uyumluluk Analiz Araçları

Araç Adı	**Yöntem**	**Analiz Seviyesi**
Lattix	DSM	Paket/Sınıf
Dclcheck	Kural Tabanlı	Paket/Sınıf
SAVE	Yansıma Modeli	Paket/Sınıf
Structure 101	Yansıma Modeli	Katman/Paket/Sınıf
Sonargraph	Yansıma Modeli	Paket/Sınıf

Bağımlılık Yapısı Matrisi Yöntemi. DSM yönteminde, kullanım bağımlılıkları bir matris ile ifade edilir. Şekil 1'de örnek bir bağımlılık yapısı matrisi gösterilmiştir. Sütunlar kullanan, satırlar ise kullanılan mimari öğeleri ifade eder. Örnekte kullanılan A ve B ifadeleri ikisi arasında kullanım bağımlılığı olabilecek herhangi bir mimari öğe olabilir (sınıf, paket, modül). Verilen örneğe göre B, A'yı kullanmaktadır.



Şekil . Örnek bağımlılık yapısı matrisi

Bu yöntemde yazılımı oluşturan sınıf veya paketler bazında kullanım ilişkileri statik analiz yöntemleri uygulanarak yazılım kaynak kodlarından çıkartılır ve şekilde gösterildiği gibi bir matris yapısında sunulur. Bu matris yapısı incelenerek istenmeyen bağımlılıklar ortaya çıkartılır, kodun daha iyi nasıl modüllere bölüneceği hakkında analizler gerçekleştirilebilir.

Bu yöntemin dezavantajı, kullanım bağımlılıkları kurallarının önceden tanımlanamaması, kodlama yapıldıktan sonra kaynak kodların analiz edilerek kural ihlallerinin matris üzerinden işaretlenmesidir.

Kural Tabanlı Yöntemler. Bağımlılık kısıtları bu alana özel olarak geliştirilen bir dil yardımıyla tanımlanabilmektedir. Böyle bir alana özel dil olan DCL dili [14] ile yapılan bazı örnek tanımlamalar aşağıda verilmiştir. Dile özel kelimeler italik yazı stiliyle vurgulanmıştır. Verilen örnekte paket adı "tr.com.aselsan.rehis.radar.B" ile

başlayan sınıfların sadece paket adı "tr.com.aselsan.rehis.radar.A" ile başlayan sınıflar tarafından kullanılabileceği belirtilmektedir.

```
module A: tr.com.aselsan.rehis.radar.A.*
module B: tr.com.aselsan.rehis.radar.B.*
only A can-access B
```

Bu yöntemin avantajı, DSM yönteminden farklı olarak kullanıcının yazılım geliştirilmeden önce mimari kısıtları ifade etmesine olanak sağlamasıdır. Bu dillerin ifade gücü oldukça yüksek olsa da görsel olmadıklarından kuralların ilk bakışta anlaşılması kolay olmamaktadır. Yöntemin çıktısı ihlal edilen kuralların ve bu duruma sebep olan yazılım kod parçalarının listelenmesi şeklinde olmaktadır.

Yansıma Modeli Yöntemi. Yansıma Modeli yöntemi [15] model tabanlıdır. İlk aşamada kullanıcının hedeflenen mimarinin bir modelini oluşturması sağlanır. İkinci aşamada bir statik analiz yazılımı yardımıyla kaynak kodun mimari modeli elde edilir. Kullanıcı tasarlanan mimari ile koddaki hangi öğelerin örtüştüğünü belirler (örtüşme modeli, İng. mapping model). Analizler araç tarafından ilk üç adımın girdileri ile gerçekleştirilir. Sonuçlar yansıma modelinde kullanıcıya sunulur. Bu yöntem mimari kısıtların yazılım geliştirilmeden önce ifade edilmesine olanak sağladığı gibi, görsel girdi ve çıktılar sağlamaktadır.

Yöntemin üç çeşit saptaması olabilmektedir:

- Uyum (İng. convergence): Üst seviye mimari modelde bulunan mimari öğenin kodda da bulunduğu anlamına gelmektedir.
- Sapma (İng. divergence): Mimari öğenin üst seviye mimari modelde bulunmadığı halde kodda gerçeklendiği anlamına gelmektedir.
- Eksiklik (İng. absence): Mimari öğe üst seviye mimari modelde tanımlandığı halde kodda gerçeklenmediği anlamına gelmektedir.

3 Radar Kullanıcı Arayüzü Yazılımları için Statik Analiz İhtiyacı

Radar alanında geliştirmekte olduğumuz kullanıcı arayüzü yazılımlarında yazılım ürün hattı yaklaşımı [16] uygulanmaktadır. Radar Kullanıcı Arayüzü (RKA) yazılımları için bir referans mimari tanımlanarak bu mimari çerçevesinde ortak varlıklar (yazılım yapıtaşları) geliştirilmiştir.

RKA Yazılımları Referans Mimarisi, OSGi (Open Services Gateway initiative) [17] çerçevesinde tanımlanan servis yönelimli (İng. service

oriented) ve olay güdümlü (İng. event driven) bir mimaridir [18]. RKA yazılımları birbirleriyle servis arayüzleri üzerinden etkileşen OSGi bileşenlerinden oluşmaktadır. Bu bileşenler arasındaki kontrol akışı olay bildirimleri üzerinden gerçekleşmektedir. Bileşenlerin bir kısmı radar alanına özel tanımlanan RADAR yazılım ürün hattından, bir kısmı da radar-elektronik harp alanına özel tanımlanan Radar Elektronik Harp Fonksiyonel Referans Modeli (REFoRM) yazılım ürün hattından [19] alınmaktadır. RKA yazılımları sadece yazılım ürün hattı varlıklarıyla oluşturulmamakta, projeye özel bileşenler de geliştirilmektedir. RKA Yazılımları Referans Mimarisi'nde bazı bileşenler proje gereksinimleri doğrultusunda eklenip çıkartılabildiklerinden opsiyonel olarak tanımlanmıştır.

RKA Yazılımları Referans Mimarisi, uyumlanabilme, tekrar kullanılabilirlik, test edilebilirlik gibi kalite faktörleri gözetilerek hazırlanmıştır. Yazılım geliştirme sürecinin başında referans mimari dokümanı temel alınarak yazılım mimarisi tanımlanmakta, temel mimari görünümleri [20] hazırlanmaktadır. Yazılım geliştirme süreci boyunca kodun mimariye uygunluğunun denetlenmesi ve uyumsuzlukların mümkün olan en erken aşamada belirlenerek daha düşük maliyetle giderilmesi hedeflenmiştir. Yazılımı oluşturan parçaların uyumlanabilme, tekrar kullanılabilirlik, test edilebilirlik gibi kalite faktörlerinin değerlendirilmesine katkıları olduğundan kullanım bağımlılıklarını tarifleyen kullanım görünümü [20] açısından referans mimariye uygunluğunun denetlenmesine öncelik verilmiştir.

RKA yazılımları en fazla 100.000 satırlık Java kaynak kodundan oluşan orta ölçekte masaüstü uygulamalardır. Yine de kod-mimari uyumsuzluğunun kod gözden geçirme gibi manüel yöntemlerle takibi maliyet-etkin olmadığından mümkün olduğunca araçlardan faydalanılması düşünülmüştür. Geliştirmekte olduğumuz projelerde kod-mimari uyumsuzluğunun yazılım geliştirme süreci içinde mümkün olan en erken aşamada belirlenebilmesi için bileşenler seviyesinde bağımlılıkların takibini sağlayan ve statik analiz yöntemleri kullanan bir aracın kullanılmasına ihtiyaç duyulmuştur.

Kodun mimariye uygunluğunun statik analizle denetlenebilmesi için aşağıda belirtilen üç temel gereksinim karşılanmalıdır:

- İstenen/hedeflenen mimarinin biçimsel (İng. formal) olarak bir yazılımın yorumlayabileceği nitelikte tanımlanabilmesi,
- Yazılım kaynak kodlarından statik analiz yöntemleriyle mimari özelliklerin bir yazılım tarafından yorumlanabilecek şekilde çıkartılabilmesi,

- Hedeflenen mimari ile gerçekleşen mimari arasındaki farkların raporlanabilmesi.

Mimariye uyumsuzluğun kodda hangi satırlardan kaynaklandığının raporlanması da tercih edilen bir yetenektir.

Kullanım bağımlılıkları açısından mimariye uygunluğun denetlenmesinde kullanılabilecek mevcut statik analiz araçları (Bkz. Tablo 1) bileşen seviyesinde değil paket veya katman seviyesinde analiz gerçekleştirmektedir. RKA yazılımlarında yeniden kullanılabilirlik paket değil OSGi bileşenleri seviyesinde sağlandığından yöntemin bileşen seviyesine uyarlanması gerekmektedir. Bunun yanı sıra varolan araçlar yazılım ürün hattı bakış açısıyla hazırlanmadığı için yazılım ürün hattından kullanılan bileşenler ile projeye özel olarak geliştirilen bileşenlerin mimari kuralların tanımlanması sırasında ayırt edilmesi mümkün olamamaktadır. Radar yazılım ürün hattı varlıklarındaki bir mimari uyumsuzluk projeye özel bileşendeki mimari uyumsuzluktan daha ciddi bir problem olarak değerlendirildiğinden, bir bileşenin projeye özel olup olmadığının mimari görünümde açıkça fark edilmesi gerekmektedir. Yapılan literatür taramasında ihtiyaçlarımıza tam olarak cevap veren bir araç bulunamamıştır.

4 RKA Yazılımlarında Yansıma Modeli Yöntemi ile Kullanım Bağımlılıklarının Denetlenmesi

Bu çalışma kapsamında, RKA Yazılımları Referans Mimarisi çerçevesinde geliştirilmiş yazılımları oluşturan bileşenler arasındaki kullanım bağımlılıklarının Yansıma Modeli yöntemi ile analiz edilmesi için bir araç geliştirilmiştir. Yansıma Modeli yöntemi, yazılım geliştirme sürecinin en erken aşamalarından itibaren kullanılabilmesi, yazılım mimari görünümünün tanımlanabilmesine olanak sağlaması ve kolay algılanabilir görsel nitelikte çıktılar üretebilmesi nedenleriyle tercih edilmiştir.

Analiz işlemi üç aşamada gerçekleştirilmektedir:

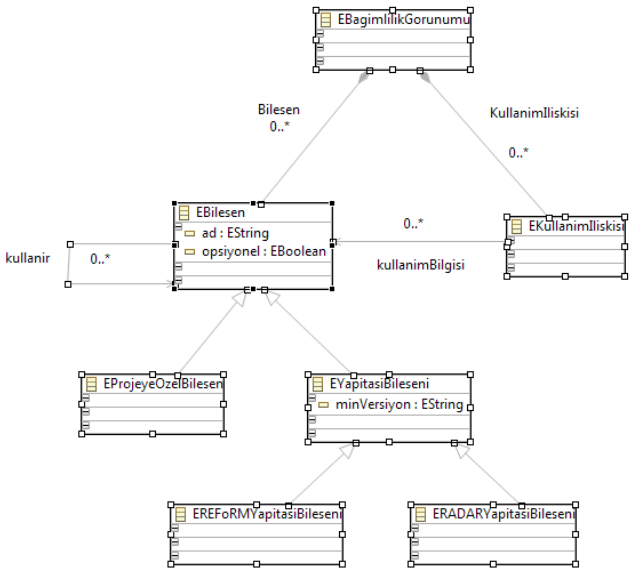
1. Yazılım mimarisinin tanımlanması: Gerçekleştirim aşamasında kullanım bağımlılıkları açısından uyulması planlanan mimari model, çalışma ortamında yer alacak OSGi bileşenleri ve bileşenler arasındaki kullanım ilişkilerini gösterecek şekilde tanımlanır.
2. Statik analiz: Çalışma ortamında bulunan OSGi bileşenlerine ait arşiv dosyaları Java programlama diliyle geliştirilen yazılımlarda kullanılabilen açık kaynak kodlu bir statik analiz aracı olan *Dependency Finder* [21] yardımıyla analiz edilerek bileşenler arasındaki kullanım ilişkileri çıkartılır.
3. Yansıma modelinin oluşturulması: Statik analiz sonucunda çıkartılan bileşen kullanım modeli ile yazılım mimari modeli karşılaştırılarak uyumlulukları, mimariden sapmaları ve eksiklikleri belirten yansıma modeli oluşturulur. Yansıma modeli bir arayüzden görüntülenir.

Bildirinin geri kalanında bu üç aşamada kullanılan yöntemler daha detaylı olarak anlatılmıştır.

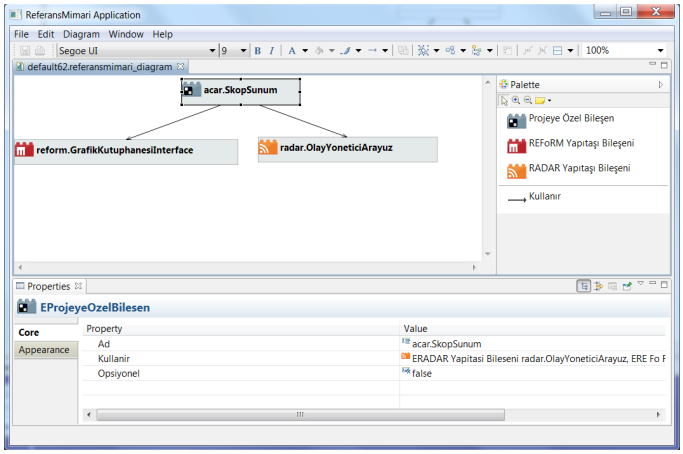
4.1 RKA Yazılımları için Kullanım Görünümü Tanımlama Aracı

Çalışmada Eclipse Modeling Framework (EMF) [22] ve Graphical Modeling Framework (GMF) [23] teknolojilerinden faydalanılarak RKA yazılımları için kullanım görünümlerinin tanımlanabileceği bir model editörü oluşturulmuştur. Model editörünün kod üretme yöntemiyle geliştirilebilmesi için tanımlanan meta-model Şekil 2'de gösterildiği gibidir. Tanımlanan meta-modele göre, kullanım görünümü (diğer adıyla bağımlılık görünümü) bileşenlerden ve aralarındaki kullanım ilişkilerinden oluşmaktadır. Farklı yazılım ürün hatları kapsamında geliştirilen yeniden kullanılabilir yazılım bileşenlerinin (yapıtaşı) farklı notasyonlarla belirtilmesine olanak vermesi için gerekli tanımlamalar meta-modelde yer almaktadır. Bileşenler projeye özel bileşen ya da yapıtaşı bileşeni tipinde olabilmektedir. Yapıtaşı bileşeni de REFoRM ya da RADAR yazılım ürün hatlarından gelen bileşenler olabilir. Meta-modele göre her bileşen için adı ve opsiyonel bir bileşen olup olmadığı tanımlanabilmektedir.

Kullanım görünümü tanımlama aracından bir örnek ekran görüntüsü Şekil 3'te yer almaktadır. Bu editör kullanılarak REFoRM ve RADAR yapıtaşı bileşenleri ile projeye özel bileşenler eklenebilmekte, birbirleri arasındaki kullanım ilişkileri tanımlanabilmektedir. Oluşturulan modeller XMI (XML Metadata Interchange) [24] formatındaki dosyalarda saklanabilmektedir.

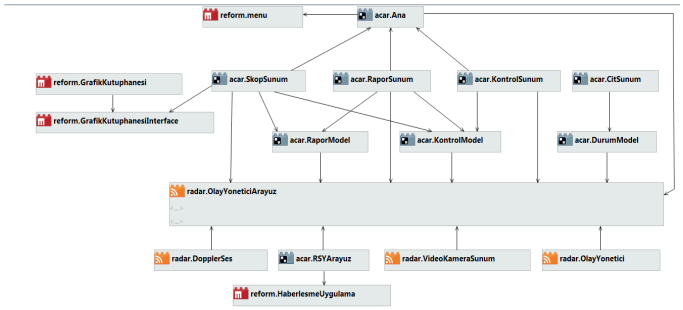


Şekil 2. RKA yazılımları için kullanım görünümü tanımlama aracı meta-modeli



Şekil 3. RKA yazılımları için kullanım görünümü tanımlama aracı ekranı

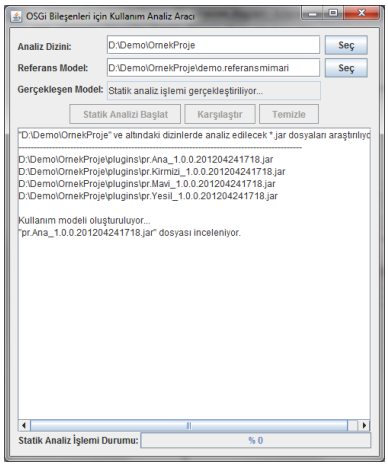
Kullanım görünümü tanımlama aracı, ACAR Kara Gözetleme Radarı Sistemi Projesi'nde, radar kullanıcı arayüzü yazılımında kullanılmıştır. ACAR kullanıcı arayüzü yazılımı da RKA Yazılımları Referans Mimarisi çerçevesinde geliştirilmekte olan bir uygulamadır. ACAR RKA yazılımı için kullanım görünümü bu araç yardımıyla Şekil 4'te gösterildiği gibi tanımlanmıştır.



Şekil 4. ACAR RKA yazılımı mimarisi kullanım görünümü

4.2 Statik Analiz

Statik analiz işleminin başlatılabilmesi ve işlemin ne aşamada olduğunun kullanıcıya raporlanabilmesi için basit bir kullanıcı arayüzü sunan (Bkz. Şekil 5) bir statik analiz yazılımı hazırlanmıştır.



Şekil 5. Statik analiz aracı kullanıcı arayüzü

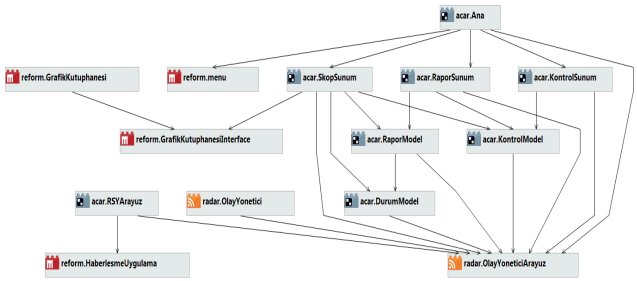
Bu yazılıma girdi olarak OSGi çalışma zamanı dizini verilmektedir. Analiz yazılımı verilen dizindeki tüm ".jar" uzantılı dosyaları (Java sanal makinesi üzerinde çalıştırılabilir Java arşiv dosyaları) bulur. Bu "jar" uzantılı dosyalar arasından bir kısmı (örneğin OSGi kullanımına ilişkin Java arşiv dosyaları), statik analiz yazılımının yapılandırma dosyasındaki bir ayarla analiz kapsamı dışında bırakılabilmektedir.

Analiz yazılımı, verilen çalışma dizinindeki analiz kapsamında olan "jar" arşivlerini açarak içindeki "Manifest.mf" dosyalarını çıkartır. Bu dosyayı yorumlayarak "jar" dosyasının bir OSGi bileşenine ait olup olmadığını, eğer OSGi bileşenine aitse diğer bileşenlerin kullanabilmesi için hangi paketleri ihraç ettiğini (İng. export) belirler. Normalde sadece yazılım kaynak kodlarının analizinden bileşen gibi üst seviye mimari öğelerin belirlenmesi mümkün değilken, “Manifest.mf” dosyalarının da analizi sayesinde yazılımda gerçeklenen bileşenlerin neler olduğu belirlenebilmiştir. Yazılım kaynak kodlarındaki paketlerden hangilerinin hangi bileşen kapsamında gerçeklenip diğer bileşenlerin kullanımına sunulduğu da aynı analizle belirlendiğinden, normalde Yansıma Modeli yönteminde analizi gerçekleştiren kişinin manüel olarak tanımlaması gereken örtüşme modelinin tanımlanması ihtiyacı ortadan kalkmıştır.

Bileşenler tarafından ihraç edilen paketlerden hangilerinin başka bileşenlerce kullanıldığının da belirlenmesi gerekmektedir. Bu amaçla Dependency Finder aracından faydalanılmıştır. Dependency Finder aracı, verilen bir jar dosyasının iç ve dış bağımlılıklarını paket/sınıf/metot seviyesinde listeleyebilmektedir. Analiz yazılımı, analiz kapsamındaki her "jar" arşivi için Dependency Finder aracını çalıştırarak paket seviyesindeki dış bağımlılıklarını çıkartır. Bu çıktılardan yola çıkarak da "Manifest.mf" dosyalarını yorumlayarak oluşturduğu üst seviye mimari modelde bileşenler arasında gerçeklenen kullanım bağımlılıklarını belirler.

Yapılan statik analiz sonucunda kodda gerçeklendiği belirlenen kullanım modeli, RKA yazılımları için kullanım görünümü tanımlama aracının tanıdığı formatta bir

XMI dosyasında saklanmaktadır. Bu nedenle gerçeklenen kullanım görünümü, kullanım görünümü tanımlama aracına yüklenerek görüntülenebilmektedir. ACAR yazılımının 0.4 sürümünde gerçeklenmiş olan kullanım görünümü Şekil 6'da gösterildiği gibidir.



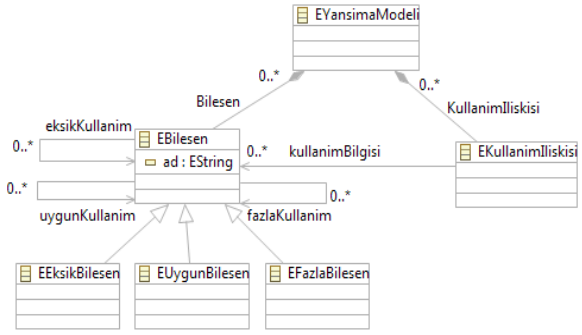
Şekil 6. ACAR yazılımı 0.4 sürümüne ait kullanım görünümü

4.3 Yansıma Modelinin Oluşturulması ve Sunumu

Şekil 5'te gösterilen statik analiz aracı kullanıcı arayüzünde "Karşılaştır" tuşuna basıldığında, statik analiz sonucunda çıkartılan kullanım görünümü ile yazılım geliştirme sürecinin başında tanımlanan mimariye ait kullanım görünümü kıyaslanmaktadır.

Karşılaştırma aşamasında, mimaride tanımlanan kullanım görünümünde bulunan her bileşen için, gerçekleştirimde bulunup bulunmadığı ile gerçekleştirimde fazladan bir bileşen bulunup bulunmadığı kontrolleri yapılır. Benzeri kontroller bileşenler arasındaki kullanım ilişkileri için de gerçekleştirilir. Sonuç olarak uyum, sapma ve eksiklik bilgilerini içeren bir yansıma modeli oluşturulur. Tanımlanan ve gerçekleşen mimari arasındaki farklılıklar statik analiz aracı tarafından metin formatında kullanıcıya raporlanır.

Yansıma modelinin grafik ortamda incelenebilmesi için yine EMF ve GMF teknolojilerinden faydalanılarak bir yansıma modeli görüntüleme aracı oluşturulmuştur. Bu aracın kod üretme yöntemiyle oluşturulabilmesi için tanımlanan meta-modeli Şekil 7'de gösterilmektedir. Buna göre yansıma modeli bileşenler ve aralarındaki kullanım ilişkilerinden oluşmaktadır. Bileşenler veya kullanım ilişkileri eksik, uygun ya da fazla olarak farklı notasyonlarla belirtilebilmektedir. Eksik bileşen ya da kullanım ilişkileri sonraki sürümlerinde yazılıma dâhil edilebileceği düşünülerek mavi renkle gösterilmiştir. Fazladan bulunan bileşen ya da kullanım ilişkileri ise sapmalara işaret ettiğinden kırmızı renkle gösterilmiştir.

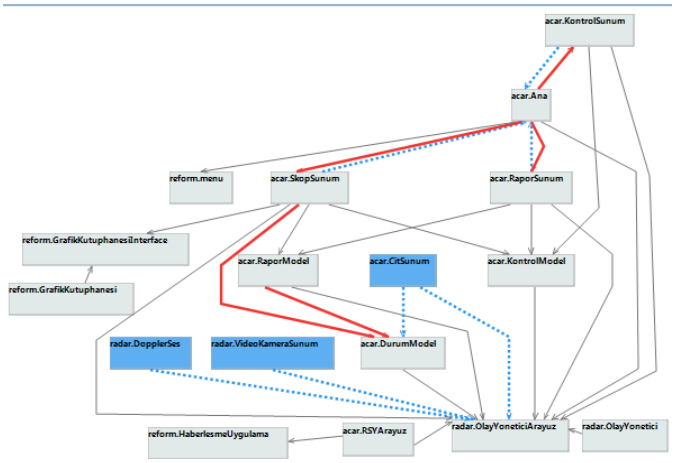


Şekil 7. Yansıma modeli görüntüleme aracı meta-modeli

Karşılaştırma aşamasında oluşturulan yansıma modeli, işlem sonunda yansıma modeli görüntüleme aracına yüklenebilecek formatta bir dosyada saklanmaktadır. ACAR yazılımının 0.4 sürümüne ait olarak çıkartılan yansıma modelinin bu araçta sunulan gösterimi Şekil 8'de gösterildiği gibidir.

5 Sonuç

Gerçekleştirilen çalışma sayesinde, RADAR yazılım ürün hattı referans mimarisi çerçevesinde geliştirilen kullanıcı arayüzü yazılımlarının kullanım bağımlılıkları açısından tasarlanan mimariye uygun olarak gerçeklenip gerçeklenmediğinin bir araç yardımı ile analiz edilebilmesi sağlanmıştır. Geliştirilen araç bir radar kullanıcı arayüzü yazılımının gerçekleştiriminde hangi aşamada olunduğunun belirlenmesine de yardımcı olmaktadır. Yazılım kodlarından kod okuma gibi maliyetli bir yöntemle bağımlılıkların çıkartılarak referans mimariye uygunluğunun denetlenmesi yerine; sadece tanımlanan kullanım görünümünün referans mimariye uygunluğunun gözle denetlenmesi ve kod-mimari uyumluluğunun ise statik analiz aracı yardımıyla denetlenmesi yeterli olmaktadır. Gelecekte uygulama mühendisliği çalışmaları kapsamında tanımlanan kullanım görünümünün RADAR alanında tanımlanan referans mimariye uygunluğunun da yazılım aracılığıyla denetlenebilmesi hedeflenmektedir.



Şekil 8. ACAR KA yazılımı 0.4 sürümü için oluşturulan yansıma modeli

Statik analiz aracı, Yansıma Modeli yöntemi kullanılarak geliştirilmiştir. RKA yazılımları OSGi çerçevesinde geliştirildiğinden, literatürde paket seviyesinde uygulanmış olan Yansıma Modeli yöntemi, OSGi çerçevesinde geliştirilmiş yeniden kullanılabilir bileşenler seviyesinde uygulanacak şekilde uyarlanmıştır. Geliştirilen araç varolan araçlardan farklı olarak sadece kaynak kodu analiz etmek yerine OSGi bileşenlerinin üst seviye tanımlarını içeren dosyaları da analiz etmektedir. Bu sayede, Yansıma Modeli yönteminin kullanıldığı araçlarda normalde kullanıcı tarafından tanımlanan örtüşme modelinin (İng. mapping model) tanımlanmasına gerek kalmamıştır.

Geliştirilen araç, Yansıma Modeli yöntemini kullanan diğer araçlardan farklı olarak kullanım görünümünde hangi yazılım öğelerinin projeye özel bileşen, hangilerinin yazılım ürün hattı bileşeni olduğunun belirtilebilmesine ve farklı yazılım ürün hatlarına ait bileşenlerin farklı notasyonlar kullanılarak gösterilebilmesine olanak sağlamıştır. Referans mimaride yazılım ürün hattından alınarak kullanılması hedeflenen bir bileşenin projeye özel ihtiyaçların çok farklılaşması nedeniyle projeye özel bir bileşen olarak geliştirilmesinin kararlaştırılması gibi durumlarda referans mimariden sapmaların daha kolay fark edilmesi sağlanmıştır. Farklı yazılım ürün hatlarından alınan bileşenlerin GMF ortamında farklı notasyonlarla gösterilebilmesi için kullanım görünümü meta-modelinin RKA yazılımlarına özel bazı tanımlamaları içermesi (REFoRM yazılım ürün hattı bileşeni ve RADAR yazılım ürün hattı bileşeni gibi özelleşmiş bileşen sınıfları) gerekmiştir. Gelecekte kullanım görünümü meta-modelinin RKA yazılımlarına özel tanımları içermeyecek şekilde genelleştirilmesi mümkün olursa araç OSGi çerçevesinde geliştirilen farklı alanlardaki yazılımlarda da kullanılabilir olacaktır.

Geliştirilen statik analiz aracı henüz prototip aşamasındadır. İlerleyen aşamalarda araçta kullanılan notasyonların Radar ve Elektronik Harp alanının mimari tasarımında kullanılmak üzere tanımlanan mimari bakış açılarına [19] uygun olması hedeflenmektedir. Aracın kullanışlılık açısından da iyileştirilebileceği yönleri

bulunmaktadır. Eclipse geliştirme ortamına entegrasyonunun sağlanması ve sürekli entegrasyon araçlarına bütünleşik çalıştırılabilmesi için bir arayüz geliştirilmesi olası iyileştirme faaliyetleri olarak değerlendirilebilir.

Bu çalışmada tekrar kullanılabilirlik ve test edilebilirlik gibi kalite faktörleri göz önüne alınarak referans mimariye uygunluğun sadece kullanım bağımlılıkları açısından ve statik olarak denetlenmesi ele alınmıştır. Radar kullanıcı arayüzü yazılımlarında önemli diğer kalite faktörleri ise güvenilirlik ve fonksiyonel doğruluktur. Bu kalite faktörleri gözetilerek referans mimaride alınan bazı önlemlerin gerçekleştirimde de sağlandığının denetlenebilmesi için yazılımın çalışma zamanındaki davranışının da analiz edilmesi gerekmektedir. Çalışmada kullanılan Yansıma Modeli yöntemi dinamik analize de uygulanabilir bir yöntemdir. Örneğin Murphy, Notkin ve Sullivan'ın çalışmasında [15], modüller arasındaki çağrımların mimari modele uygun olarak gerçekleştiği bu yöntemle denetlenmiştir. Olay güdümlü bir referans mimari çerçevesinde geliştirmekte olduğumuz radar kullanıcı arayüzü yazılımlarında çeşitli olayların referans mimari dokümanında verilen sıralama diyagramlarına uygun olarak gerçekleştiği yine Yansıma Modeli yöntemi ile denetlenebilir. Bu yolla, kullanıcı arayüzünde yapılan herhangi bir ayarın radara gönderilmesinde bir aksaklık olmadığı veya radarda geçerli olan çalışma ayarlarıyla radar kullanıcı arayüzünde sunulan ayarların bağlantı kopup gelmesi gibi hata durumlarında da tutarlılığının sağlandığı garantilenmiş olacaktır.

Teşekkür

Değerli yorumlarıyla çalışmaya katkıları bulunan Özgü Özköse Erdoğan, Baki Demirel, Gökhan Kahraman, Ülkü Şencan ve Reyhan Ergün'e teşekkürlerimi sunarım. GMF aracının kullanımı konusunda yardımları için Gökhan Kahraman'a ayrıca teşekkürlerimi sunarım.

Kaynaklar

1. P.L. Dewayne, A.L. Wolf. Foundations for the Study of Software Architecture, ACM SIGSOFT, Software Engineering Notes vol. 17, no 4, pp. 40-42, Oct 1992.
2. M. Shaw, P. Clements. The Golden Age of Software Architecture: A Comprehensive Survey, http://www.cs.cmu.edu/~Compose/GoldenAge_TRv6.pdf, 2006.
3. Teknik borç teriminin tanımı, http://martinfowler.com/bliki/TechnicalDept.html
4. M. Dalgarno, When Good Architecture Goes Bad, http://www.methodsandtools.com/archive/archive.php?id=85
5. J. Knodel, D. Popescu. A Comparison of Static Architecture Compliance Checking Approaches, Proceedings of the Working IEEE/FIP Conference on Software Architecture, pp. 44, 2007.
6. L. Passos, R. T. M. T. Valente, R. Diniz, N. Mendoça. Static Architecture-Conformance Checking: An Illustrative Overview, IEEE Software, pp. 82-89, September/October 2010.
7. Lattix aracı ana sayfası, http://www.lattix.com
8. DCL belirtimi ana sayfası, http://java.llp.dcc.ufmg.br/dclsuite
9. Structure101 aracı ana sayfası, http://structure101.com
10. Sonargraph aracı ana sayfası, http://www.hello2morrow.com/products/sonargraph

11. SAVE aracı ana sayfası, http://fc-md.umd.edu/save
12. M. Lindvall, D. Muthig. Bridging the Software Architecture Gap, Software Technologies, pp.92-96, June 2008.
13. Sullivan K., Cai Y., Hallen B., Griswold W. The Structure and Value of Modularity in Software Design, Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2001.
14. R. T. M. T. Valente. A Dependency Constraint Language to Manage Object-Oriented Software Architecture, Software Practice and Experience, 39, pp. 1073- 1094, 2009.
15. G. C. Murphy, D. Notkin, K. J. Sullivan. Software Reflexion Models: Bridging the Gap between Design and Implementation, IEEE Transactions on Software Engineering, Vol. 27, No. 4, April 2001, pp. 364-380.
16. P. Clements, L. Northrop. Software Product Lines: Practices and Patterns. Boston, MA:Addison-Wesley, 2002.
17. The OSGi Alliance, OSGi Service Platform Enterprise Specification, Release 4 Version 4.2, March 2010.
18. E. Çilden, G. Bayraktar, G. Işık, B. Demirel, OSGi Çerçevesinde Bir Olay Güdümlü Mimari Uygulaması, Sixth National Conference on Software Engineering (UYMS 2012), Ankara, Türkiye, Mayıs, 2012.
19. B. Tekinerdogan, Ö. Özköse Erdoğan, O. Aktuğ, Çoklu Ürün Hatları için Hazırlanan Mimari Bakış Açıları ile Radar ve Elektronik Harp Alanının Mimari Tasarımı, Proceedings of the Fifth National Conference on Software Engineering (UYMS 2011), pp. 115-123, Ankara, Turkey, September, 2011.
20. P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. Documenting Software Architectures Views and Beyond, 2nd Edition, 2011.
21. “Dependency Finder” aracı ana sayfası, http://depfind.sourceforge.net/
22. “Eclipse Modeling Framework” ana sayfası, http://www.eclipse.org/modeling/emf/
23. “Graphical Modeling Project” ana sayfası, http://www.eclipse.org/modeling/gmp
24. XMI belirtimi ana sayfası, http://www.omg.org/spec/XMI/