Askeri bir gömülü yazılımın bileşen tabanlı bir mimari kullanılarak refaktör edilmesi

Berkhan Deniz[1], Gökhan Öztaş[1], Soner Çınar[1]

[1]Gömülü ve Gerçek Zamanlı Yazılım Tasarım Müdürlüğü, SST Sektör Bşk.

ASELSAN A.Ş.

{berkhand, goztas, scinar}@aselsan.com.tr

Özet. Gömülü yazılımlar, aynı fiziki birimler kullanıldığı halde, sistem gereksinimlerinin güncellenmesi, birimlerin değişik şekillerde konfigüre edilmesi gibi sebeplerle benzer sistemlerde dahi farklılaşabilmektedir. Bu durum yazılımların karmaşıklaşmasına, yeniden kullanımın azalmasına ve harcanan işçilik miktarlarının artmasına sebep olmaktadır. Bu çalışmada, teslim edilmiş bir askeri gömülü yazılımın, benzer projelerde sistematik bir şekilde yeniden kullanımını sağlamak amacıyla bileşen tabanlı ve farklı seviyelerde konfigüre edilebilir bir mimari kullanılarak refaktör edilmesi anlatılacaktır. Ardından, mimari evrim sürecinde ve sonrasında karşılaşılan zorluklar ve bulunan çözümler aktarılacaktır.

Anahtar Kelimeler: Bileşen tabanlı yazılım geliştirme, refaktör, gömülü yazılım geliştirme

Abstract. Embedded systems may differentiate even if they are comprised of identical physical units; due to the facts that requirements are updated and units are configured differently. This situation complicates the software, reduces the reuse rates and causes enhancement of total effort. In this study, we will explain refactoring of a military embedded software using a component-based architecture with multi-level configuration infrastructure. Then, questions encountered throughout and after this architectural evolution and solutions offered will be reported.

Keywords: Component based software development, refactoring, embedded software development

1 Giriş

Bilindiği üzere Aselsan ülkemizin en önde gelen askeri sistem çözümleri üreten firmasıdır. Geçtiğimiz on beş yılda Aselsan tarafından üretilen sistemlerin hem çeşitliliği hem de karmaşıklığı oldukça artmıştır. Ancak bu duruma zıt bir şekilde sistemlerin geliştirme ve teslimat süreleri her geçen gün azalmaktadır.

Bu gelişmelere paralel olarak, geliştirilen sistemleri yöneten gömülü sistem yazılımları da her yeni projeyle birlikte daha da karmaşıklaşmakta ve de bu yazılımların geliştirilmesi için ayrılan süreler azalmaktadır. Yazılımların geliştirilme süresini ve maliyetini azaltmak için en etkili metotlardan birinin yeniden kullanımı artırmak olduğu bilinmektedir. Bileşen tabanlı yazılım mühendisliği, bileşenleri bir araya getirerek sistem yazılımlarının geliştirilmesinin sağlanması olarak tanımlanmaktadır. Ancak, bileşenlerin geliştirilmesi ve birleştirilmesi süreçlerinin önceden belirlenmiş kurallarla, yani bir mimari ile yönetilmesi gerekmektedir.

Bu çalışmada, teslim edilmiş bir askeri gömülü yazılımın, benzer projelerde sistematik bir şekilde yeniden kullanımını sağlamak amacıyla bileşen tabanlı ve farklı seviyelerde konfigüre edilebilir bir mimari kullanılarak refaktör edilmesi anlatılacaktır.

Çalışmanın geri kalanı su şekilde düzenlenmiştir: bölüm 2'de bileşen tabanlı yazılım mühendisliği ile ilgili literatür çalışmaları özetlenmiştir. Bölüm 3'te bileşen tabanlı mimari öncesi geliştirilen yazılım mimarileri, bölüm 4'te geliştirilen bileşen tabanlı mimari yaklaşımı anlatılmıştır. Son olarak, sonuçlar ve gelecek için çalışma önerileri bölüm 5'te verilmiştir.

2 Literatür

Gelişen ve değişen teknolojiler sayesinde yazılım geliştirme için uygulanan metotlar zamanla geleneksel yöntemleri bırakarak önce nesne tabanlı, şimdi de bileşen tabanlı yazılım mühendisliğine doğru evrilmiştir. Bileşenleri bir araya getirerek sistemlerin geliştirilmesini sağlamak, bileşenlerin yeniden kullanılabilir birimler olarak üretilmesini sağlamak, sistemlerin bileşenlerin değiştirilmesiyle güncellenmesini ve bakımının yapılabilmesini sağlamak bileşen tabanlı yazılım mühendisliğinin en temel hedefleri olarak gösterilmektedir [10]. Diğer bir deyişle; bileşen tabanlı yazılım mühendisliği, hazırda olan daha küçük ürünleri –bileşenleri-, standart ya da standart benzeri bileşen modelleri kullanarak yazılım üretme disiplini olarak tanımlanabilir [2].

Lahon ve Sharma ilgili çalışmalarında bileşen tabanlı yazılım mimarisinin etkilerini farklı alanlardaki kullanımlarını da gözeterek özetlemişlerdir [7]. Onlara göre yeniden kullanılabilirlik kavramı bileşen tabanlı yazılım mühendisliğinin belkemiğidir ve

yapısal tasarımın ve nesne tabanlı paradigmaların başarısının ardından, yazılım geliştirme alanında bir sonraki devrim olarak görülmektedir.

Literatürde bileşen tabanlı yazılım geliştirmenin en temel avantajları olarak geliştirme zaman ve maliyetinin azalması, kalite, özelleşme ve uzmanlaşmanın artması, birbiri ile uyumlu çalışabilen yazılımlar üretme olanağı gibi etkiler sıralanmaktadır [1, 2, 5].

Bileşen tabanlı yazılım geliştirmenin en önemli ayırt edici özelliği bileşenlerin sadece hâlihazırda karşılaması gereken gereksinimlere göre değil, daha genel ve kapsamlı gereksinimlere göre geliştirilmesidir. Böylelikle bileşenin farklı birimlerle de uyumlu olabilmesi hedeflenmektedir [1].

Bileşen tabanlı geliştirmenin bazı riskleri de bulunmaktadır. Crnkovic çalışmasında; yeniden kullanılabilir bileşen geliştirilmesinin yeniden kullanılmayan bileşenlere göre 3 ila 5 kat daha fazla emek gerektirdiğini belirtir [5]. Fakat diğer bir çalışmada ise yeniden kullanılabilir bileşenin tasarım ve dokümantasyon gibi işçiliklerle beraber beşinci yeniden kullanımdan sonra kendi maliyetini çıkarabildiği tespit edilmiştir [8]. Diğer bir risk; yeniden kullanılabilir bileşenlerin, yazılımların yeniden kullanılmayan bölümlerine göre çok daha fazla değişikliğe maruz kalabilmesi olarak görülmektedir [9].

Frank, Crnkovic ve Runeson yaptıkları olay çalışmasında; küresel bir şirketin dünya üzerindeki farklı merkezlerde geliştirilen yazılım ürünlerinin birbiriyle uyumlu çalışabilmesi için bileşen tabanlı bir yazılım mimarisine geçmelerini gözlemlemiş ve analiz etmişlerdir [2]. Çalışmanın genel amacı bileşen oluşturma surecini gözlemlemek ve bileşen tabanlı mimarinin faydalarını değerlendirmek olarak tanımlanabilir. Bu çalışmaya göre bileşen tabanlı mimari dağıtık yazılım geliştirmeyi etkili bir şekilde desteklemektedir. Ayrıca, gereken işçilik miktarının da oldukça azaldığı gözlemlenmiştir. Tüm bunların yanında, sistemin performans ve çalışma zamanı ile ilgili diğer kalite metriklerinin yeterli seviyede kaldığı gözlemlenmiştir. Diğer taraftan bileşen tabanlı mimariye geçiş çalışmalarının temel zorluklarından biri de yeni platformu yeterince genel, esnek ve genişleyebilir yapmak olmuştur. Diğer bir kısıt da, zamanlamalar gibi önemli gerçek zaman gereksinimlerinin yeni eklenen yazılım mimari seviyelerinden eklenmemesi gerekliliğidir. Fakat genel olarak bileşen tabanlı mimariye geçiş çalışmaları sayesinde daha modüler ve daha iyi dokümante edilmiş bir sistem elde edilmiştir. Bu iki özelliğin genelde kaliteyi artırdığı bilinmektedir [2]. Bu gözlemler, Szyperski'nin görüşleriyle de uyumludur: Bileşen tabanlı mimari yaklaşımı modülerliği sağlamak için kullanılabilir. Yeni ürünler geliştirilmesi için gereken işçilik sürelerinin azalması da çoğunlukla, önceden tekrardan yapılan bazı parçaların geliştirilen çerçeve sayesinde yeniden kullanımının sağlanması sayesinde olmaktadır. Bu durumun ayrıca yazılım kalitesini iyileştirdiği de bilinmektedir, çünkü tekrardan yazılan kısımlarla birlikte yeni hatalara sebep olunması da engellenmektedir [4].

Daha önce birçok alanda bileşen tabanlı mimariye geçiş çalışmaları yapılmıştır [3]. Bu çalışmalar göstermiştir ki bileşenler sayesinde hem ürünlerin idame edilebilirlikleri artarken hem de ayni zamanda performans gereksinimlerinin de limitler içerisinde kalabilmektedir.

Bileşen tabanlı yazılım geliştirme ile sağlanması öngörülen kazanımlar, yazılım endüstrisinin diğer alanlarında olduğu kadar gömülü sistemler alanında da oldukça ilgi çekicidir [1, 6]. Geleneksel olarak, gömülü sistemler için yazılım geliştirilirken birinci öncelik donanım kaynaklarını en az şekilde harcamak için performans metriklerinde verimliliği en fazlaya çıkartmak olmaktadır. Donanım fiyatlarındaki sürekli azalmanın ve gömülü sistemlerin karmaşıklığının sürekli artmasının bir sonucu olarak, idame edilebilirlik, güvenilirlik ve güvenlik gibi özelliklerin önemi de gittikçe artmaktadır [2]. Gömülü sistemler alanında, bileşen tabanlı yazılım geliştirme, üretim maliyetlerini azaltmak ve sistemlerin idame edilebilirliğini artırmak için önemli bir araç olarak gösterilmektedir [4]. Diğer taraftan, kalite, güvenilirlik ve performans gibi fonksiyonel olmayan gereksinimlerin sağlanması gömülü yazılımlar için çok daha kritiktir. Hazır bileşenler kullanılarak yeni uygulamalar geliştirildiğinde, sadece fonksiyonel olarak doğru çalışmasını sağlamak değil aynı zamanda fonksiyonel olmayan özelliklerinin de doğru çalışmasını sağlamak gerekmektedir [1]. Bütün yazılım sistemleri fonksiyonel ve fonksiyonel olmayan özellikler içermektedir. Ancak, gömülü sistemler geliştirilirken fonksiyonel olmayan gereksinimler son ürün için kritiktir ve sistemin doğruluğu açısından bilişim sistemlerine göre çok daha önemlidir [1].

3 Bileşen tabanlı mimari öncesi geliştirilen yazılım mimarileri

Bu bölümde bileşen tabanlı mimari çalışmaları öncesindeki dönemde Aselsan Gömülü ve Gerçek Zamanlı Yazılım Tasarım Müdürlüğü bünyesindeki Hava Savunma Silah Sistemleri ve Görüntü İşleme (HSSS-Gİ) ekibi tarafından geliştirilen gömülü silah sistemleri yazılım mimarileri anlatılacaktır.

HSSS-Gİ ekibi projelerini gerçek zamanlı bir işletim sisteminde C++ dilini ve UML tabanlı bir yazılım geliştirme aracı kullanarak gerçekleştirmişlerdir. Ekip içerisinde -üyeler ve ekibin kişi sayısı değişmekle beraber- yaklaşık 6-8 yazılım mühendisi çalıştığı söylenebilir.

3.1 Birinci nesil yazılım mimarisi

Ekip tarafından geliştirilen birinci nesil yazılımda katmanlı yazılım mimarisi kullanılmıştır. Bu mimaride, ana yönetim işlevlerinin ve senaryoların gerçeklendiği katmana Kontrol katmanı, alt sistemlere ait operasyonel işlemlerin yapıldığı katmana

Operasyon katmanı, donanım üzerinde çalışan birimlerin bulunduğu katmana Birim katmanı ve donanıma erişim için kullanılan katmana da Donanım katmanı adı verilmiştir.

Birinci nesil yazılımların tasarım ve geliştirme çalışmaları, yazılım gerekleri yatay ve dikey düzlemlerde çözümlenerek ele alınmış ve bu katmanlı yapı kullanılan UML tabanlı geliştirme aracı ile geliştirilen yazılımda doğrudan uygulanmıştır. Bu şekilde, UML modelleme süreci uygulanarak sistem tasarımı problem tabanına indirgenerek tasarım çıktılarının anlaşılabilirliği, yeniden kullanılabilirliği, esnekliği ve taşınabilirliği hedeflenmiştir (Bkz. Şekil 1).

Bu mimaride, Operasyon katmanı ile Birim katmanı arasında ve Birim katmanı ile Donanım katmanı arasındaki haberleşme soyutlama arayüzleri üzerinden gerçeklenmiştir. Yazılımın, birim değişikliklerinden etkilenmemesi hedeflenmiştir.

Birinci nesil mimarinin hedefleri bugün için de doğruluğu kabul edilen hedeflerdir. Ancak uygulama aşamasında beklenilen fayda görülememiştir. Bu durumun sebepleri takvim sıkışıklığı, iletişim eksikliği, tanımlanmış bir yeniden kullanım altyapısının bulunmaması, farklılaşan ihtiyaçların iyi yönetilememesi, işletim sistemine ve işlemci destek yazılımlarına bağımlı kod geliştirilmesi, bileşen kullanım kuralları tanımlanmamasından dolayı birbirine sıkı sıkıya bağlı bileşenlerin ortaya çıkması, acil ihtiyaçlara yönelik yapılan değişikliklerle esnekliğin ortadan kalkması olarak sıralanabilir.

Şekil 1. Birinci nesil katmanlı mimari ve katmanlar arası bilgi akışı

2000-2005 yılları arasında geliştirilen bu mimari ile bir proje geliştirilmiştir.

3.2 İkinci nesil yazılım mimarisi

Birinci nesil mimaride oluşan yukarıda bahsettiğimiz sıkıntılardan dolayı ekip tarafından ikinci nesil yazılım mimarisi geliştirilmiştir. Bu yazılım, önceki mimariye kıyasla önemli iyileştirmeler içermektedir. Temel olarak aynı hedefler amaç edinilmiştir.

Birinci nesil yazılımın donanım katmanında bulunan tüm bileşenler bu yazılımda yeniden kullanılmıştır. Ancak diğer tüm bileşenler yeniden yazılmıştır.
Birinci nesil yazılım mimarisinde Operasyon katmanı ve Kontrol katmanı arasındaki karar mekanizması çok iyi belirlenemediğinden, kontrollerin bazıları Operasyon katmanında bazıları Kontrol katmanında bazıları da her iki katmanda birden yer almaya başlamıştır. Bu tür sebeplerden dolayı, ikinci nesil yazılım mimarisinde Kontrol katmanı kaldırılmıştır (Bkz. Şekil 2).

Şekil 2. İkinci nesil katmanlı mimari ve katmanlar arası bilgi akışı

Bu mimari bazı açılardan bileşen tabanlı bir mimariye benzemektedir. Çünkü bazı bileşenler ortak bir arayüzden kalıtımlanmıştır ve birbirinin yerine kullanılabilmektedirler, ancak bu sistematik bir çalışmanın sonucu olarak değil de benzer işi yapan farklı fiziki birimlerin kontrol modüllerinin bileşen olarak değerlendirilebilmesinin doğal bir sonucudur.
İkinci nesil mimaride geliştirilen yazılım bloklarının neredeyse tamamı kabuklanmak suretiyle bileşen tabanlı mimaride kullanılabilmiştir.
2005-2008 yılları arasında geliştirilen bu mimari ile dört proje geliştirilmiştir.

4 Bileşen tabanlı mimari yaklaşımı

Birinci ve ikinci nesil mimari ile üretilen yazılımların koştuğu sistemler silah sistemleri alanında Aselsan tarafından geliştirilen ilk sistemlerdir. Bu sistemlerin başarısının ardından hem bu sistemlere çok benzer hem de oldukça farklı birçok yeni proje geliştirme faaliyetleri başlamıştır. Bu projelerin takvimleri de öncekilere nazaran giderek kısalmıştır. Ancak kullanılan mimari sistematik yeniden kullanımı sağlama amacıyla geliştirilmediği için, yeni geliştirilecek projelerin ihtiyaçlarına yanıt vermesi mümkün olmamıştır. Bu sebeple hem proje hem de bileşen seviyesinde

yeniden kullanımı gözeten yeni bir mimari ihtiyacı geliştirilmesine karar verilmiştir. Bu bölümde IRIS (İng: Intelligent Recognition Identification Systems) adını verdiğimiz bileşen tabanlı mimari anlatılacaktır.

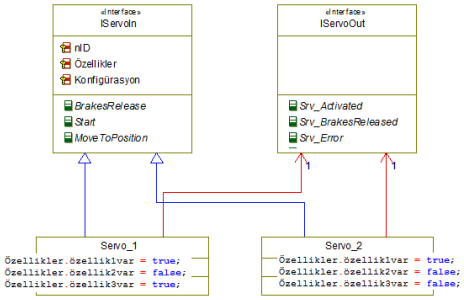
4.1 Bileşen tabanlı mimariye geçiş çalışmaları

Bileşen tabanlı mimariye geçiş için yapılan ilk iyileştirme donanım seviyesi bileşenlerinin soyutlama sınıfları ile arayüzlerinin ortaklanması olmuştur. Böylelikle birçok farklı haberleşme protokolü ve donanıma özgü yetenek aynı arayüz sınıfları üzerinden erişilebilir olmuştur. Bu çalışma sonucunda Donanım katmanı Platformdan Bağımsızlık Bileşeni olarak güncellenmiştir.

Bileşen tabanlı mimariye geçiş yolunda gerçekleştirilen diğer hamle de birim bileşen arayüzlerinin ortaklanması çalışması olmuştur. Bu çalışma kapsamında, önceden geliştirilmiş projelerde kullanılan tüm birim tipleri için güncel arayüzler temel alınarak alan analizi çalışması yapılmıştır. (Birim bileşeni ve birim tipi kavramları karışabilmektedir. Kamera bir birim bileşeniyken, herhangi bir marka kamera bir birim tipidir.) "X bileşeninden beklentiler nelerdir, geçmişteki projelerde X birimleri hangi arayüzleri sağlamıştır, beş-on yıllık yol haritasında geliştirilecek ya da geliştirilmesi olası projelerde ne tür arayüzlere ihtiyaç olacaktır" gibi sorular sorulmuştur. Bu çalışma sonucunda bazı yeteneklerin ilgili bileşenin alan analizinde bulunmasına rağmen tüm birimler tarafından sağlanmamış olduğu tespit edilmiştir. Bu durum, aynı bileşenin tüm birimlerinde ortak olmayan yeteneklerin "Özellikler" sınıfı altında toplanmasıyla çözülmüştür. Tüm birimlerin yapıcısında o birim için Özellikler sınıfı doldurulmaktadır. Her birimin arayüzü altında kullanıma açılan özellikler sınıfı sayesinde, kullanıcı sınıfların birimler arasındaki bu tip farklılıkların farkında olmaları sağlanmıştır. Bu sayede kontrol sınıflarının birim tipinden bağımsız olarak geliştirilmesi sağlanmıştır. Bu sınıflar ilgili birimin özelliklerine bakarak ona özgü senaryoları koşturmaktadır.

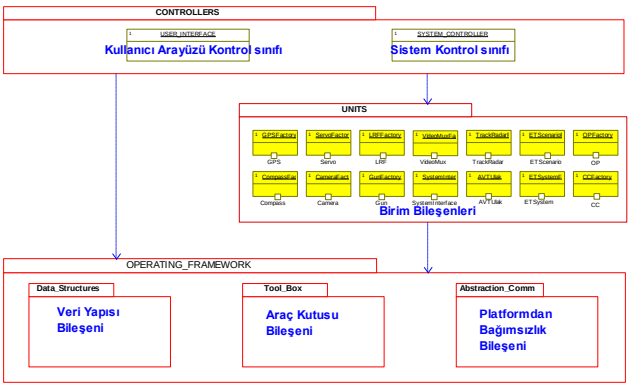
Benzer şekilde birimlerin bazı yeteneklerinin de farklı projelerde değişkenlik gösterebildiği tespit edilmiştir. "Özellikler"den farklı olarak bu yetenekler birime özgü değil de projeye özgü değişkenlikleri içermektedir. Bu durum projeye özgü değişkenliklerin "Konfigürasyon" sınıfı altında toplanmasıyla çözülmüştür. Sonraki bölümlerde daha ayrıntılı anlatılacak bu durum sayesinde projeye özgü değişikliklerden birimlerin etkilenmemesi sağlanmıştır.

Arayüzlerin ortaklanması ve Özellikler ve Konfigürasyon sınıflarının oluşturulmasıyla bileşen kavramı son halini almıştır. İki farklı Servo birimi için örnek bir bileşen yapısı (ortaklanmış arayüzler ve özelliklerin doldurulması) Şekil 3'te gösterilmiştir.



Şekil 3. Örnek bileşen mimarisi

Bu aşamadan sonra bileşen tabanlı mimariye doğru ilerlemeler daha üst seviyedeki düzenlemelerle devam etmiştir. İkinci nesil mimaride Operasyon katmanı içindeki alt sistemler arasında birbirine benzer kontrollerin yapıldığı gözlenmiştir. Bu benzerliklerin ortaklanması amacıyla, Operasyon katmanı altındaki Kullanıcı Arayüzü kontrol sınıfı haricindeki tüm alt sistem yetenekleri birleştirilerek tek bir Sistem kontrol sınıfı altında toplanmıştır. İki temel parça altında –Kullanıcı Arayüzü ve Sistem kontrol sınıfları- birleştirilen eski Operasyon katmanı çok daha kolay yönetilebilir bir yapıya dönüşmüştür. Bileşen tabanlı mimarinin genel görünümü Şekil 4'te gösterilmiştir.



Şekil 4. Bileşen tabanlı mimarinin (IRIS) genel görünümü

4.2 Konfigürasyon yaklaşımı

Bir önceki bölümde anlatıldığı üzere, bileşenler ve kontrol sınıflarıyla ilgili alt yapılar geliştirildikten sonra sıra sistematik bir konfigürasyon yaklaşımı oluşturulmasına gelmiştir. Şekil 5'te gösterildiği gibi konfigürasyon yönetimi için katmanlı bir yapı benimsenmiştir. Platformdan bağımsızlık seviyesindeki konfigürasyon ile yazılımın üzerinde koşacağı platform ve işletim sisteminin belirlenmesi ve üst seviye bileşenlerin bu seviyedeki değişikliklerden etkilenmemesi sağlanır.

Şekil 5. Katmanlı konfigürasyon yaklaşımı

Birim seviyesindeki konfigürasyon sayesinde hangi tipteki birimlerin yaratılacağı belirlenir. Yaratma esnasında birime ait konfigürasyonlar (Bkz. Bölüm 4.1 – Konfigürasyon sınıfı) birime iletilir. Birim aldığı bu konfigürasyon ile kendini ilkler, gerekiyorsa başka bileşenlerin yaratılmasını aynı yöntemle sağlar. Bir sonraki bölümde birim bileşenlerinin yaratılması daha ayrıntılı olarak açıklanacaktır.

Bileşen tabanlı mimariye göre, bu mimari kullanılarak üretilen tüm sistemlerde Sistem Kontrol sınıfının bulunması zorunludur. Bu seviyedeki konfigürasyon sayesinde ise sistemin mod ve durumları ve sistem senaryoları yapılandırılır. Farklı projelerdeki senaryosal değişiklikler buradaki konfigürasyonda yapılan güncellemeler ile yönetilir.

Bu mimari kullanılarak üretilen sistemlerde Kullanıcı Arayüzü kontrol sınıfının bulunması zorunlu değildir. Bu sebeple bu seviyedeki konfigürasyon bu sınıfın yaratılıp yaratılmayacağını ve eğer yaratılacaksa hangi tipte bir kullanıcı arayüzü sınıfının gerçekleneceğini belirler.

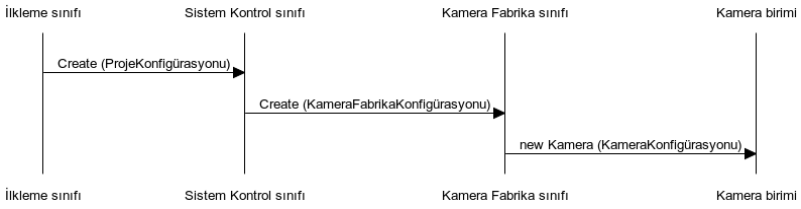
4.3 Uygulanan bileşen tabanlı mimaride bileşen

Geliştirilen tüm projelerde ortak olarak kullanılması hedeflenmiş, anlamlı bir bütün oluşturan modüler yapıdaki yazılım parçaları "bileşen" olarak tanımlanmıştır.

Şekil 4'teki mimari görünümde de belirtildiği gibi en alttaki İşletim Çerçevesi katmanında Veri Yapısı, Araç Kutusu ve Platformdan Bağımsızlık bileşenleri bulunmaktadır. Veri Yapısı bileşeni yazılımın genelinde ortak olarak kullanılan veri yapılarının tanımlandığı bileşendir. Yeniden kullanılabilir tüm veri yapıları, silah

sistemleri ve görüntü işleme alanına özel veri yapıları bu bileşende toplanmıştır. Araç Kutusu bileşeninde de bu alanlara özgü her türlü yeniden kullanılabilir yardımcı sınıflar toplanmıştır. Platformdan bağımsızlık bileşeni de yazılım genelinin platformdan ve işletim sisteminden bağımsız çalışmasını sağlayan bileşendir.
Şekil 4'teki mimari görünümün ikinci seviyesinde Birim bileşenleri bulunmaktadır. Bileşen tabanlı mimari ile geliştirilen silah sistemleri yazılımlarında kullanılabilecek tüm birim arayüzleri ilgili birim bileşeni içerisinde tutulmaktadır. Devam eden dört alt başlıkta birim bileşenlerinin diğer özellik ve yetenekleri anlatılacaktır.

Bileşenler nasıl yaratılır? Birim bileşenlerinde Fabrika şablonu kullanılmaktadır. Bu şablon, konfigürasyon ile belirlenmiş özelliklere göre yapılandırılmış birime ait objeyi arayüz tipindeki kullanıcı yazılıma (Sistem Kontrol sınıfı, Bkz. Şekil 4) sunar. Sistem Kontrol sınıfı hiçbir seviyede birim kodlarına ait başlık dosyalarını görmemektedir. Tüm yaratma ve ilkleme işlemleri bileşen sorumluluğundadır.



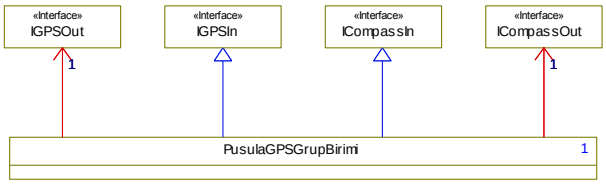
Şekil 6. Birim bileşeni yaratılma ardıl işlemi

Şekil 6'da örnek bir bileşen yaratılma ardıl işlemi gösterilmiştir. Proje konfigürasyonu İlkleme sınıfı tarafından çalışma zamanında okunduktan sonra, Sistem Kontrol sınıfı bu konfigürasyon ile ilklenir. Ardından, bu konfigürasyonun içindeki bileşenlere özgü konfigürasyonlarla tüm bileşen fabrikaları ilklenir. Bileşen fabrikaları ise yaratılacak birimleri onlara ait konfigürasyonla yaratır. Bu sayede çalışma zamanında, yaratılması gereken tüm birim bileşenlerinin o projeye özgü birim konfigürasyonları ile yaratılma işlemi tamamlanmış olur.

Bileşen tipleri. Gömülü yazılım birim bileşenleri genellikle fiziksel birimlerle haberleşmeyi sağlayan yazılım parçalarından oluşmaktadır. Geliştirilen mimaride de benzer şekilde tüm birim bileşenleri ayrı bir fiziksel birim tipini gerçeklemek üzere yaratılmıştır: Kamera, Servo, Pusula, GPS, Lazer Mesafe Bulucu gibi. Özetlemek

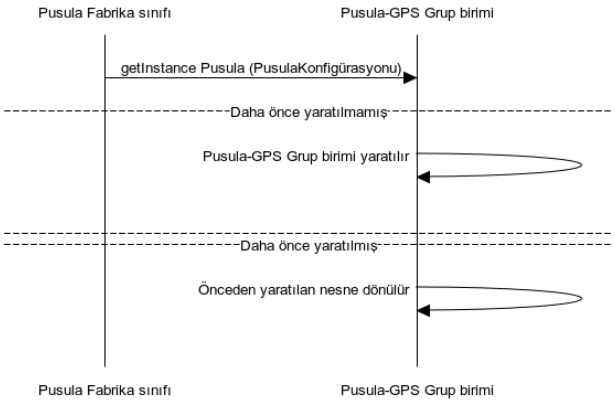
gerekirse, birim bileşenleri ortaklanmış bileşen arayüzlerinden (Bkz. Şekil 3) sadece birini gerçekleyecek şekilde tasarlanmıştır.

Bu yaklaşım birçok farklı projede yüzlerce farklı birim için sorunsuz olarak çalışabilmiştir. Ancak, aynı fiziksel birim içerisinde aynı haberleşme arayüzünü kullanan birden fazla birim tipinin bulunduğu birimlerin kullanıma alınmasıyla bu yaklaşımı güncellemek gerekmiştir. Örnek vermek gerekirse: Elektro-Optik grubu (Kamera + Lazer), Navigasyon grubu (Pusula + GPS), Pan-Tilt grubu (Kamera + Servo) gibi.



Şekil 7. Pusula-GPS Grup bileşeni sınıf diyagramı

Grup bileşeni kavramıyla bu sorun çözümlenmiştir. Şekil 7'de Pusula-GPS Grup bileşeninin sınıf diyagramı gösterilmiştir. Grup bileşenlerinde birden fazla birim arayüzü gerçeklenir. Proje konfigürasyonunda Grup bileşeni bulunuyorsa, Sistem Kontrol sınıfı Şekil 8'deki ardıl işleme uygun olarak bu bileşenleri yaratmaktadır. Grup birimlerinde tekillik şablonu uygulanmıştır. Bu sebeple, bir projede grup bileşenini oluşturan birimlerin ikisi birden kullanılıyorsa Grup birimi yalnızca biri için yaratılmaktadır.



Şekil 8. Grup bileşeni yaratılma ardıl işlemi

Gömülü yazılım birim bileşenleriyle ilgili zaman içerisinde ortaya çıkan ve çözülmesi gereken bir diğer sorun da tek bir görevi birden fazla bileşenin yerine getirmesi gerektiğinde ortaya çıkmıştır. Bu durumun ilk örneği X ve Y eksenlerinde pozisyon belirlemek için kullanılan Servo bileşeninde X ekseninin ve Y ekseninin aynı birim değil "farklı" birimler tarafından belirlenmesi zorunluluğudur. Diğer örnek ise kuzey, yanca ve yalpa yönlerinde yönelim açılarını belirlemek için kullanılan Pusula bileşeninde kuzey açısının ve yanca ile yalpa açılarının farklı birimler tarafından belirlenmesi durumudur.

Bileşke bileşen kavramıyla bu sorun çözümlenmiştir. Şekil 9'da Bileşke Servo bileşeni için ardıl işlemi verilen bu yöntemde, Bileşen Fabrika sınıfına Bileşke birimlerin tipleri ve konfigürasyonları girilir. Fabrika sınıfı bileşke birimi bu konfigürasyonlar ile yaratır. Bileşke birim de tekrar Fabrika sınıfına ilgili birimleri yarattırır. Bu yöntem sadece basit bir Bileşke birimin hazırlanmasını gerektirir, diğer yaratma işlemleri Fabrika sınıfı tarafından yapılmaktadır.

Şekil 9. Bileşke bileşen yaratılma ardıl işlemi

Bu bölümde açıklanan Grup bileşeni ve Bileşke bileşen kavramları sayesinde gömülü sistemlerde karşılaştığımız en karmaşık birim kombinasyonları dahi bileşen tabanlı mimari bozulmadan çözümlenebilmektedir. Şöyle ki, GPS ve Pusula verilerini ölçen bir birimden GPS ve kuzey açılarını, başka bir birimden de yanca ve yalpa açılarını almamız gerektiği bir projede iç içe Bileşke bileşen ve Grup bileşeni kullanarak, sadece konfigürasyon değişiklikleriyle çözüm üretebilmekteyiz.

Bileşen arayüzleri ne zaman güncellenir? Bileşen arayüzleri alan analizi yapılarak oluşturulmasına karşın bileşen havuzuna eklenen yeni birimlerle ve değişen proje gereksinimleriyle beraber bu arayüzlerin güncellenme ihtiyacı doğabilmektedir. Ancak arayüzler doğrudan güncellenmez, ekip üyeleriyle ve diğer paydaşlarla beraber (sistem ve proje mühendisleri) bu ihtiyacın gerekliliği sorgulanır. Gerekiyorsa, bileşenin Özellikler sınıfı da güncellenerek idame edilmekte olan projelerin etkilenmemesi de sağlandıktan sonra değişiklik yapılır. Ancak mimarinin katı

olmaması, değişkenliğe açık olması sebebiyle geçmiş projelerin güncellenmesi ihtiyacının göze alınabildiği durumlar da olabilmektedir.

4.4 Geliştirilen mimari ile sağlanan kazanımlar

Geliştirilen bileşen tabanlı mimari sayesinde işletim sistemi ve platformu, içerdiği birim tipleri ve kombinasyonları birbirinden farklı onlarca gömülü sistem yazılımı sadece konfigürasyon değişiklikleri ile sistematik olarak üretilebilmektedir. Konfigürasyon alt yapısı sayesinde derlemeden sistem davranışı değiştirilebilmekte ve daha önceden kullanılmış her türlü birim, yazılım güncellemesine gerek duyulmadan konfigürasyon ile bağlanabilmektedir. Mimarinin hiçbir seviyede statik, katı olmaması sayesinde birbirinden çok farklı ister kolaylıkla karşılanabilmektedir.
Bileşen tabanlı mimariye ilk geçiş sürecinde teslim edilen projelerde yapılan testler, sistem performansında kabul edilemez kayıplar olmadığını göstermiştir. Bu sayede yapılan refaktör çalışmaları devam edebilmiş ve son halini alabilmiştir.
Son olarak, yeni geliştirilen sistemler için üretilen yazılımlara ayrılan işçilik sürelerinde astronomik denebilecek boyutlarda azalmalar gözlenmiştir. Bileşen tabanlı mimarinin geliştirilme sürecindeki pilot projelerde dahi işçilik miktarında kayda değer azalmalar gözlemlenmesine rağmen, mimari sonlandıktan sonraki işçilik değerlerinin ikinci nesil mimariye göre bile 1/10 civarında olduğu ölçülmektedir.
2008 yılı sonrasında geliştirilen IRIS mimarisi ile otuzun üzerinde geliştirilmiş ve geliştirilmesi devam eden proje bulunmaktadır.

5 Sonuç ve gelecek için planlar

Bu çalışmada Aselsan Gömülü ve Gerçek Zamanlı Yazılım Tasarım Müdürlüğü bünyesindeki Hava Savunma Silah Sistemleri ve Görüntü İşleme ekibi tarafından geliştirilen gömülü silah sistemleri yazılım mimarileri anlatılmıştır. Yapılan refaktör çalışmalarıyla bileşen tabanlı mimariye doğru yaşanan evrim süreci gösterilmiştir. Bileşen tabanlı mimarinin sistematik olarak uygulama örnekleri gösterilmiş ve elde edilen kazanımlar özetlenmiştir.
Bundan sonraki süreçte geliştirilecek yeni projelere ve değişen ihtiyaçlara göre alan analiz çalışmalarının sürmesi planlanmaktadır. Bu kapsamda, önceki sistemlerde bulunmayan birim ve bileşenlerin mimariye eklenmesiyle mimarinin genişlemeye devam edeceği düşünülmektedir.

Kaynaklar

Atkinson, Colin, et al. "Component-based software development for embedded systems–an introduction." Component-Based Software Development for Embedded Systems. Springer Berlin Heidelberg, 2005. 1-7.

Lüders, Frank, Ivica Crnkovic, and Per Runeson. "Adopting a component-based software architecture for an industrial control system–a case study."Component-Based Software Development for Embedded Systems. Springer Berlin Heidelberg, 2005. 232-248.

Algestam, H., Offesson, M., Lundberg, L.: Using Components to Increase Maintainability in a Large Telecommunication System. In: Proceedings of the Ninth Asia-Pacific Software Engineering Conference (2002) 65–73.

Szyperski, C.: Component Software – Beyond Object-Oriented Programming. 2nd edition. Addison-Wesley, Reading, MA (2002).

Crnkovic, Ivica. "Component-based software engineering—new challenges in software development." Software Focus 2.4 (2001): 127-133.

Galla, Thomas M., et al. "Refactoring an automotive embedded software stack using the component-based paradigm." *Industrial Embedded Systems, 2007. SIES'07. International Symposium on*. IEEE, 2007.

Lahon et al. "Component Based Software Engineering – At a Glance" *International Journal of Advanced Research in Computer Science and Software Engineering* 4.9 (2014): 605-609.

Mrva M. Reuse Factors in Embedded Systems Design. High-Level Design Techniques Dept. at Siemens AG, Munich, Germany, 1997.

Crnkovic I, Larsson M. A Case Study: Demands on Component-based Development. Proceedings 22nd International Conference on Software Engineering, ACM Press, 2000.

Heineman G, Councill W. Component-based Software Engineering, Putting the Pieces Together. Addison Wesley, 2001.