Gömülü Kontrol Yazılımlarında İşlev Katmanının Referans Model Oluşturmak için Yeniden Yapılandırılması Yaklaşımı

Gökhan Kahraman[1]

[1] Radar Elektronik Harp ve İstihbarat Sistemleri (REHİS) Grubu, ASELSAN A.Ş., Ankara

[1]e-posta: gokhank@aselsan.com.tr

Özetçe

Bu bildiride ASELSAN Radar ve Elektronik Harp İstihbarat Sistemleri (REHİS) Görev Yazılımları Müdürlüğü (GYM) kapsamında geliştirilen bir projede, Gömülü Kontrol Yazılımları Alan Katmanında Referans Model Oluşturmak için gerçekleştirilen yeniden yapılandırma süreci sunulmakta ve değerlendirilmektedir.

Bu çalışmada, kullanımda olan mevcut [1] Referans Mimari başlangıç noktası olarak alınmıştır. Zaman olarak daha önceden çalışmaları başlamış olan bir Yazılım Konfigürasyon Birimi'nin (YKB) Alan katmanında yer alan işlev yapısı, tasarımı yeni başlamış ikinci YKB'nin de ortak olarak kullanabileceği yeniden kullanılabilir yapılar haline getirilmiştir. Bu sayede farklı görevleri olan iki yazılım için, benzer işlev yapıları kullanılarak, işlev katmanında referans bir model oluşturulmuştur.

1. Giriş

Yazılım Mimarileri geçtiğimiz son on yılda güçlü araştırmaların yapıldığı bir konu olarak ortaya çıkmıştır [2]. Mimarilerin tanımlanması ve analizi için birçok sayıda yaklaşım önerilmiştir [2]. Ancak bu yaklaşımlar genellikle geleneksel masa-üstü uygulamalar için ortak varsayımlardan yola çıkmışlardır. Önerilen mimari tabanlı çözümler Gömülü Yazılımlar özelinde ele alındığında ise bazı zorluklarla karşı karşıya kalınmaktadır. Bu yüzden geleneksel mimari tasarım yaklaşımlarını, gömülü kontrol yazılımları özelinde kullanarak referans model oluşturmak özel bir dikkat gerektirmektedir.

Gömülü kontrol yazılımları bir sistemde yer alan birimlerin, işleyiş senaryolarını gerçekleştiren, donanım birimlerini kontrol eden, sistemde yer alan diğer gömülü yazılımların grafiksel kullanıcı arayüzü gibi uygulama yazılımları ile arasındaki eşlemeyi sağlayan yazılımlardır.

ASELSAN REHİS GYM'de Gömülü Kontrol Yazılımları mimarisi çatısı oluşturulmuş ve tasarlanmıştır [1]. Mimaride katmanlar, katmanlar arası haberleşme yöntemleri ve katmanların görevleri tanımlanmıştır. Bu katmanlardan İşlev Katmanı olarak adlandırılan katman alana özel işlevlerin gerçekleştiği katmandır, mimari kapsamında sadece alt katmanlarla haberleşme yöntemi verilmiştir, belirlenen yöntemler ile her projenin ve hatta Yazılım Konfigürasyon Biriminin (YKB) ihtiyacına göre projeye özel geliştirilmektedir.

Gömülü kontrol yazılımları işlevlerinin tasarımı zaman kısıtı gibi sebeplerden dolayı, proje odaklı geliştirildiğinde YKB içinde büyük yapılar haline dönüşebilmekte ve işlevler arasında istenmeyen bağımlıklar kurulabilmektedir. Bu durumda bir sınıfı ya da paketi yeniden kullanmak zorlaşmaktadır. Ortaya çıkan zorluklar bu yapıları yeniden kullanmak isteyen tasarımcıda bir sınıfı ya da paketi yeniden kullanmak yerine yeniden tasarlamak gibi bir eğilim ortaya çıkarmaktadır.

REHİS GYM kapsamında geliştirilen bir projede, bir alt sistemin yazılım tasarımı yapılırken farklı bir alt sistem yazılımında benzer işlevlerin kullanıldığı görülmüştür. Bu işlevlerin ortak olarak kullanılmasının yazılımın test edilmesi, güvenilirliğinin artması, kodun bakımının yapılabilmesi ve işgücünden kazanç sağlanması gibi faydalar sağlayacağı düşünülerek referans bir model oluşturulması kararı verilmiştir. Bu amaçla işlev katmanındaki ortak kullanılacak işlevlerin yeniden yapılandırılması için bir takvim belirlenmiş ve uygulanmıştır.

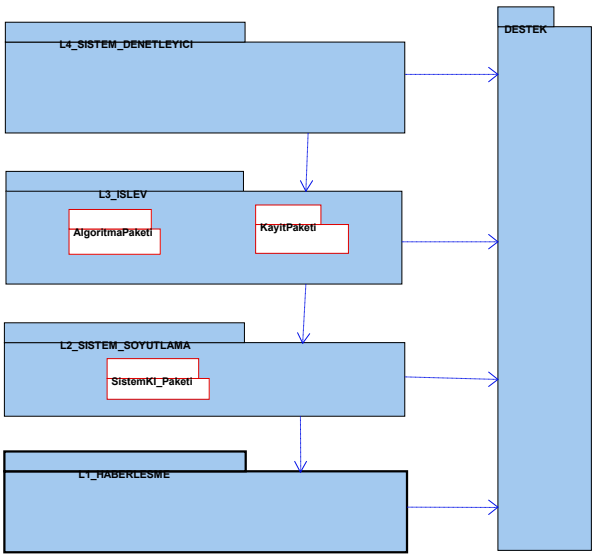
Bu çalışma gerçekleştirilirken literatürde yer alan yazılım kalıplarından faydalanılmıştır ve bu kalıpların gömülü sistemlerde kullanımı ile ilgili modeller oluşturulmuştur.

Bu bildiride anlatılan çalışmalar referans mimari ve yeniden yapılandırma konularında eksiksiz ya da nihai olarak düşünülmemelidir, daha çok gelecekteki tartışmalar için bir başlangıç noktası olarak ele alınmalıdır.

Bildirinin geri kalanında sunum şu şekildedir: 2. Bölüm Gömülü Kontrol Yazılımları Mimari Yapısını açıklamaktadır. 3. Bölüm İşlev Katmanı'nın Mimari'deki yerini vurgulamakta ve bu katmanda Referans Model ihtiyacının ortaya çıkmasındaki süreci anlatmaktadır. 4. Bölüm İşlev Katmanının Referans Model için yeniden yapılandırılması sürecinde geçilen aşamaları anlatmaktadır. 5. Bölümde İşlev Katmanında Referans model oluşturmak için kullanılan yöntemler ve yapılar verilmektedir. 6. Bölümde İşlev Katmanı Referans Modeli için örnek bir yapı sunulmaktadır. 7. Bölüm yapılan çalışmayla elde edilen kazanımları değerlendirmektedir. 8. Bölüm yapılan çalışma sürecinde karşılaşılan zorluklara değinmekte ve dikkat edilmesi gereken konuları vurgulamaktadır. 9. Bölüm çalışmayı, bundan sonra hedeflenen çalışmalarla birlikte değerlendirmektedir.

2. Gömülü Kontrol Yazılımları Mimari Yapısı

Gömülü Kontrol yazılımlarında katmanlı mimari yapı kullanılmaktadır. Katmanlı mimari, tasarım sonucu oluşan paketleri soyutlama seviyelerine göre belli bir hiyerarşik yapıda organize etmektedir [3].



Şekil 1: Katmanlar Arası İlişkiler

2.1. Katmanlar

Şekil 1'de Gömülü Kontrol Yazılımlarında kullanılan katmanlı mimari yapı gösterilmiştir. Gömülü Kontrol Yazılımları için dört katmanlı bir mimari belirlenmiştir. Bu katmanlar Haberleşme, Sistem Soyutlama, İşlev ve Sistem Denetimi katmanlarıdır. Bu bildiride işlev katmanına yoğunlaşılacağı için yazılımın yapısının daha iyi anlaşılabilmesi için diğer katmanlardan kısaca bahsedilecektir. Bu bölümde anlatılan katmanların kullanımı ile ilgili detaylı bilgi için [1] incelenebilir.

Haberleşme Katmanı: Sistemdeki birimlerle haberleşmekte kullanılan arayüzleri gerçekleyen sınıfların bulunduğu katmandır. Bu katman REHİS GYM bünyesinde kütüphane olarak geliştirilmiş ve projelerde referans olarak kullanıma alınmıştır.

Sistem Soyutlama Katmanı: Kontrol yazılımının haberleştiği birimlerle arasında giden gelen mesajların işlendiği seviyedir.

Sistem Denetleyici:. Sistem denetleyici, sistem soyutlama katmanında birbirinden soyutlanmış olan birimler arasında ve bu birimlerin işlev sınıfları ile arasındaki iletişimi yönlendirir.

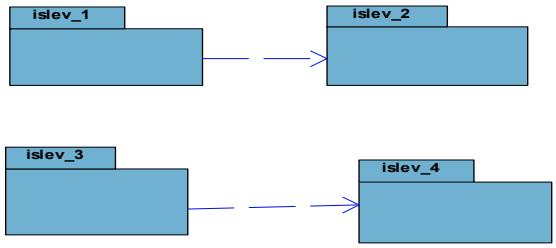
İşlev Katmanı: Kontrol yazılımının işlevsel sorumluluklarının yerine getirildiği bir katmandır. Kontrol yazılımın bir senaryo dahilinde yönettiği işlevler bu katmanda yer alan aktif sınıflarca yönetilir.

3. Mimari'de İşlev Katmanı

Sistem birimleri arası eşlemeyi yönetmek İşlev katmanındaki sınıfların temel görevlerinden biridir. Bu açıdan İşlev katmanında yer alan sınıflar sistem senaryolarına doğrudan bağımlıdır. Sistemdeki işlevler, birbirini etkileyen yapıda olabileceğinden, diğer katmanlardan farklı olarak bu katmanda yer alan paketler ve sınıflar arası ilişkiler kurulabilir[1].

İşlev katmanında kurulabilen ilişkiler ve bağımlılıklar için koyulan kurala uyulurken göz önünde bulundurulması gereken önemli noktalar vardır, çünkü işlev yapıları arasında kurulan bağımlılıklar işlevlerin karmaşıklığı arttıkça yönetmesi zor bir hale gelmektedir.

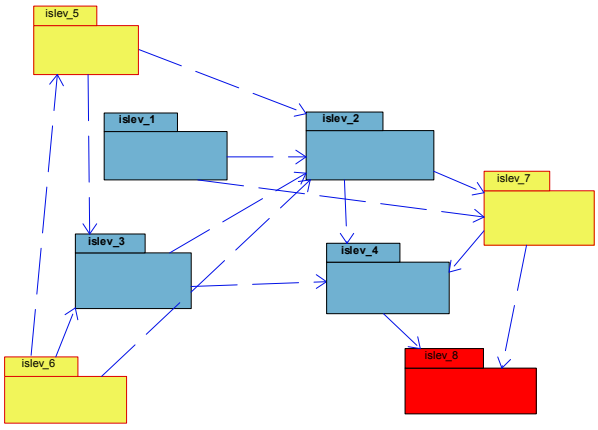
Örneğin başlangıçta işlev yapıları arasında Şekil 2'deki gibi bir ilişki olabilir.



Şekil 2: Başlangıçta İşlev Katmanı Bağımlılıkları

Bu modelde, tasarım yeni başladığı için açıkça tanımlanmış birkaç tane paket vardır. Ve birbirleriyle olan ilişkileri anlaşılır seviyededir.

Fakat ilerleyen zamanlarda tasarım büyümeye başladığında ve yeni işlevler koda eklendikçe işlevler arasındaki bağımlılıklar artmakta ve tasarımın yapısı Şekil 3'e benzemeye başlamaktadır.



Şekil 3: İlerleyen Zamanlarda İşlev Katmanı Bağımlılıkları

Tasarıma yeni eklenen işlevlerde yeniden kullanılabilirlik ve performans etkileri düşünülmekte fakat proje odaklı

çalışmalarda zaman baskısı altında kaçınılmaz olarak yukarıdaki şekildeki gibi bir yapı oluşabilmektedir. Çünkü bu gibi durumlarda işlevin gerçeklenmesi sırasında odak noktası uzun vadede oluşabilecek etkiler olmamaktadır.

İşlevler arasındaki bağımlılıklar arttıkça kodda yapılacak küçük bir değişiklik büyük hatalara yol açabilmektedir. Bu yüzden uygun zaman ve takvim ele geçirildiğinde kodda yeniden yapılandırma yapılmalıdır.

İşlev katmanında gerçekleştirilecek bir yeniden yapılandırmada, tasarım içindeki hata olasılığının azaltılmasının yanı sıra tasarım kalıpları kullanılarak yeniden kullanılabilirliği artırmak ve referans model oluşturmak ile sağlanabilecek faydalar da düşünülmelidir.

Referans model ile fayda sağlanabilecek durumlara örnekler şunlar olabilir:

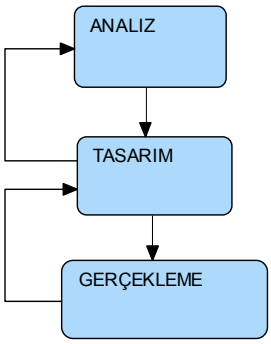
- Gömülü Kontrol yazılımlarında Algoritma yapılarının oluşturulması ve kullanılması birden fazla projede değerlendirilebilir bir işlevdir.
- İşlev modellerinde birden fazla işlev yapısı tarafından kullanılabilecek bir değişkenin kullanımının belirlenen bir yöntemle ortaklanması sağlanabilir.
- Daha önceden test edilmiş ve kullanılmış bir yapının koda eklenmesi için yöntemlerin oluşturulması ve kuralların belirlenmesi sağlanabilir.

Bundan sonraki bölümlerde Gömülü Kontrol yazılımlarında gerçekleştirilen işlevlerden bazılarının ortaklanması için kullanılan strateji ve yöntemlerden örnekler anlatılacaktır.

4. İşlev Katmanının Referans Model için Yeniden Yapılandırılması Aşamaları

İşlev Katmanında referans model oluşturma sürecinde yapılanlar 3 ana aşamadan oluşmaktadır.

1. **Analiz:** İşlev yapıları analiz edilmiştir. Yeniden kullanılabilir yapılar belirlenmiştir. Bu yapıların diğer sınıflarla olan bağımlılıkları ve karmaşıklıkları incelenmiştir ve bağımlılık haritası oluşturulmuştur.
2. **Tasarım:** Yeniden Kullanılabilir yapılar içinde kurulabilecek tasarım yöntemleri belirlenmiştir. Bu işlem için literatürde bulunan tasarım kalıpları incelenmiştir. Bu tasarım kalıpları işlev katmanındaki yapıların oluşturulmasında kullanılmıştır.
3. **Gerçekleme:** Tasarımda gerekli değişiklikler ve düzenlemeler yapılmıştır. Değişikliklerin ardından testler gerçekleştirilmiştir.



Şekil 4: Yeniden Yapılandırma Aşamaları

Şekil 4'te Yeniden yapılandırma aşamaları gösterilmiştir. Bu aşamalar arasındaki geçişler tek yönlü değildir. Bir önceki aşamaya geri dönüş yapılarak, yinelemeli bir yöntem izlenebilir.

5. İşlev Katmanında Referans Model Oluşturmak İçin Kullanılan Yöntemler

Yeniden kullanılabilirlik övgüye değer bir amaçtır, ama zordur[4]. İşlevlerde olabilecek küçük bir değişiklik yeniden kullanılmak istenen yapıda değişiklik yapma ihtiyacını ortaya çıkarabilir. Bu yüzden işlev katmanında yeniden kullanılabilir somut yapılar oluşturmak yerine, yeniden kullanılabilir referans modeller oluşturmak ve bu modellerin kullanım yöntemlerini, girdi ve çıktılarını ve ilişkilerini tanımlamak gereklidir.

Farklı işlevler için oluşturulan Referans Modelde kullanılan yöntemler aşağıda anlatılmaktadır. Bu yöntemlerin literatürde adı geçen tasarım kalıpları ile benzerlikleri dolayısıyla ilgili oldukları tasarım kalıpları da referans olarak verilmektedir. Tasarım Kalıplarının farklı amaçlarla kullanımları için [4], [5] referansları incelenebilir.

5.1. Ortak Bilgi Dağıtıcısı Yapısı

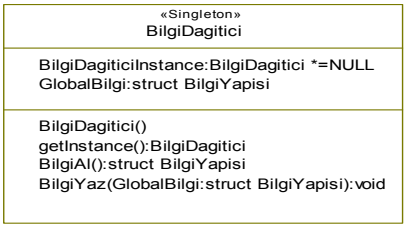
İşlev katmanında sorumlulukları farklı olan fakat aynı değişkene ihtiyacı olan yapılar bulunabilir. Bu durumda literatürde "Singleton"[4] olarak adlandırılan tasarım kalıbı kullanılabilir. Bu yapıda, farklı işlevlerdeki farklı nesneler bir bilgiye erişmek istediğinde bu tasarım kalıbı kullanılarak oluşturulan nesnenin (BilgiDağıtıcı) örneğine (instance) erişir ve kullanmak istediği bilgiyi alır. Bu işlem için aşağıdaki komutu kullanır.

```
struct BilgiYapisi Bilgi = itsBilgiDagitici-
>getInstance()->BilgiAl();
```

Bu nesne içindeki bilgiyi güncellemek isteyen işlev yapısı ise, bu nesnenin içindeki bilgiyi yine bu nesneden bir örnek alarak günceller. Bu işlem için aşağıdaki komutu kullanır.

```
itsBilgiDagitici->getInstance()-
>BilgiYaz(struct BilgiYapisi);
```

Şekil 5'te Ortak Bilgi Dağıtıcısı Yapısı için örnek bir sınıf gösterilmiştir.



Şekil 5: Ortak Bilgi Dağıtıcı Yapısı

BilgiDağıtıcı nesnesinden sadece bir tane yaratılması, "Singleton" tasarım kalıbı kullanılarak garanti altına alınır.

Bu yapı sayesinde işlev yapılarının arasındaki bağımlılık azaltılmış ve bağımsız işlev yapıları oluşturmak için olanak sağlanmış olur. BilgiDağıtıcısı'nın kullanılmadığı durumda ortak değişken işlev katmanındaki bir sınıf içinde yer alacaktır ve bu değişkene ihtiyacı olan sınıf ile ilişki kurmak gerekecektir. Bu istenmeyen bir durumdur. Çünkü sorumlulukları farklı olan iki yapının birbirleriyle bağımlılığının kurulması sonucunda bu yapılardan birinin yeniden kullanılmak istenmesi durumunda diğer nesneye ihtiyaç duyulacaktır.

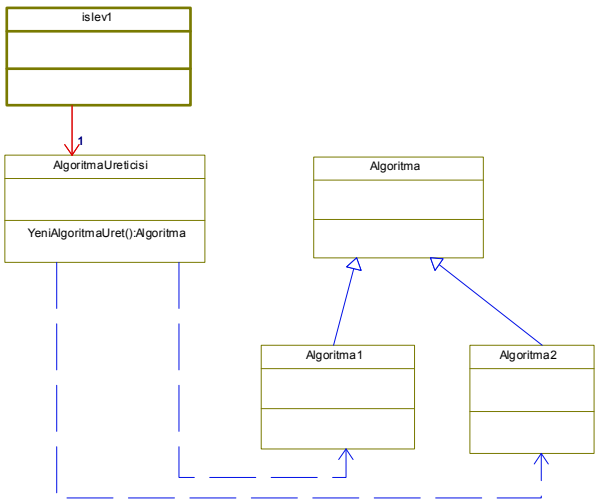
Yukarıda anlatılanların yanı sıra BilgiDağıtıcı sınıfı, aynı anda bilgi erişiminin önüne geçilmesi istenen durumlarda da kullanılabilir. Bu sınıf içinde semaphore yapısı kullanılarak ortak değişkene erişim kontrol altına alınabilir. Bilgi üzerindeki kontrol BilgiDağıtıcı sınıfına aittir.

5.2. Algoritma Üretimi Yapısı

Gömülü Kontrol Yazılımlarının grafik verisi hazırlama ya da algoritma çalıştırma gibi görevleri vardır. Gömülü kontrol yazılımları sistem için merkezi bir görevde olduğu için, alt birimlerden aldığı veriyi anlaşılabilir ve amaca yönelik bir işlemden geçirmesi gereken durumlar sıklıkla ortaya çıkar. Bu durumlarda dinamik olarak nesne yaratma ya da öldürme gibi işlevlerin gerçekleştirilmesi gerekir. Bu görevler için işlev Katmanı'nda referans model ile yeniden kullanılabilir yapılar oluşturabilmek amacıyla literatürde "Abstract Factory"[4] olarak adlandırılan tasarım kalıbı kullanılır. Bu yapının kontrol yazılımları özelinde örnek bir kullanımı ve yapısı aşağıda anlatılmıştır.

Bu yapıda farklı görevleri olan Algoritma sınıfları ile bağımlılık Algoritma Üretici'sine verilir. Algoritma Üreticisi ile Algoritma sınıfları arasında bağımlılık vardır. Bir işlev bir Algoritma sınıfına ihtiyacı olduğunda bu nesneyi Algoritma sınıfı'ndan ister. Algoritma üreticisi ilgili nesnenin örneğini isteği yapan işlev sınıfına döndürür. İşlev sınıfı aldığı algoritma nesnesini kullanır.

Algoritma Üretimi Yapısı için örnek sınıf diyagramı Şekil 6'da verilmiştir.

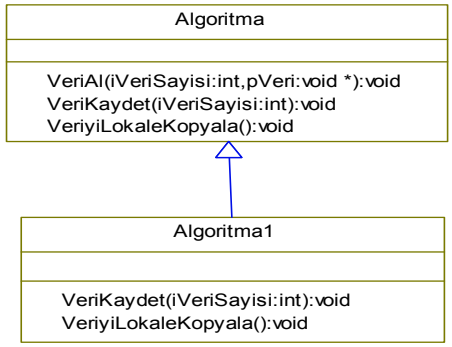


Şekil 6: Algoritma Üretimi Yapısı

İşlev katmanında Algoritma üretimi için kurulan yapı yeniden kullanılabilir Algoritma işlevleri oluşturmak için önemli bir avantaj sağlamaktadır. Farklı sorumluluklar yüklenen bir İşlev sınıfı bir Algoritma nesnesi'ne ihtiyacı olduğunda algoritma üretimi için Algoritma Üreticisi'ni kullanır. Bu şekilde işlev sınıfı ve Algoritma sınıfları bağımsız olarak yeniden kullanılabilir yapılar haline getirilmiş olur. Referans modelde, Algoritma Üretimi Yapısı kullanılarak oluşturulmuş modeller bulundurularak, tasarımcının tasarım yaparken belirlenmiş bu kurallara uyması sağlanır.

5.3. Algoritma Şablon Yapısı

Referans Model oluşturulurken Algoritma çalıştırma görevi için ortak bir yapı kurmak önemlidir. Algoritma çalıştırılırken gerçekleştirilen adımlar incelendiğinde benzer aşamalardan geçtikleri görülür. Bu ortaklık Referans Modelde bir araya getirilir. Bu işlem için "Template"[4] olarak isimlendirilen tasarım kalıbı kullanılır.



Şekil 7: Algoritma Üretimi Yapısı

Şekil 7'deki örnek sınıf diyagramında Algoritma Şablon Yapısı verilmiştir. Ana sınıfta tanımlı fonksiyonlar "virtual" olarak tanımlanmıştır. VeriAl(…) fonksiyonunun içeriği aşağıdaki gibidir;

```
VeriyiLokaleKopyala(pVeri, iVeriSayisi);
if (iVeriSayisi > 0)
{
  VeriKaydet(iVeriSayisi);
}
```

Burada VeriAl(…) şablon fonksiyondur. Bir algoritma sınıfında Veri Alma işlevinin nasıl yapılacağını tanımlar. Bu kod parçasında veri öncelikle Algoritma nesnesinin içine kopyalanır, sonra da veri Kayıt edilmek üzere ilgili işleve aktarılır. Şablon fonksiyonu'nun içinden çağrılan fonksiyonlar çengel fonksiyonlar olarak isimlendirilir. Şablon yapısı ile oluşturulan Algoritma sınıfını gerçekleyen çocuk nesneler çengel fonksiyonlarını kendi işlevleri doğrultusunda gerçekler. Çünkü bu fonksiyonlar içinde gerçekleştirilenler farklı algoritma işlevleri için değişebilir yöntemlerdir.

Ana Algoritma sınıfı, kullandığı şablon yapısı ile birlikte Referans Modele uygun hale getirilmiş olur. Referans Model'den alınan bu yapı kullanılarak istenilen işleve göre gerçek algoritma işlevleri gerçeklenir. Bu yapı sayesinde farklı görevler için gerçeklenen Algoritma yapıları, bu göreve ihtiyacı olan bir yazılım tarafından modeline eklenerek kullanılabilir.

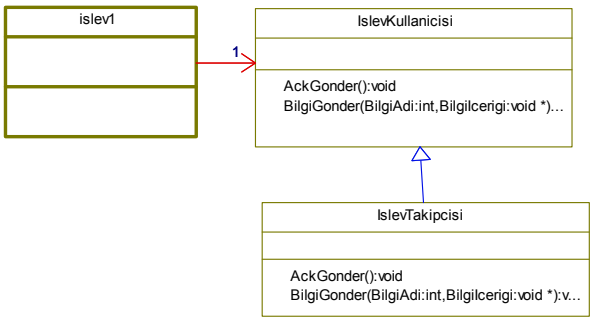
Aynı algoritma görevini yerine getirecek farklı 2 yazılım için şablon yapısı kritiktir. Çünkü bu yapı, bir yazılım için oluşturulmuş Algoritma sınıfının diğer yazılım tarafından kullanılmasına olanak sağlar.

5.4. İşlev Takipçi Yapısı

Referans Model kullanımına uygun yeniden kullanılabilir yapılar oluşturmak için en önemli yapılardan biri İşlev Takipçi Yapısıdır. Bu yapı literatürde "Observer" olarak adlandırılan tasarım kalıbı uygulamalarından biridir.

İşlev yapıları belirli sorumluluklarla oluşturulmuş sınıf ya da sınıflardan oluşur. Kullanımları sırasında dışarıdan erişim için arayüz sağlarlar, farklı bir işlevden bilgiye ihtiyaç duyarlar ya da diğer bir işleve bilgi gönderirler. . Bu durumlarda diğer bir işlev ile bağımlılık kurmak gerekmektedir. Referans Model'den kullanım için bu yöntem uygun değildir çünkü bir işlevi kullanmak istediğimizde diğerini de modelimizde kullanmamız gerekecektir. Bunun önüne geçmek için İşlev Takipçi yapısı kullanılır.

İşlev Takipçi Yapısı işlevlerin diğer işlevlerle olan ilişkilerinin bir araya toplandığı yapıdır. Bunun için işlev yapısı içinde IslevKullanıcısı isminde bir sınıf tanımlanır. İlgili işlevin IslevKullanıcısı ile ilişkisi vardır. Dışarı bilgi göndermek istediğinde ya da bilgi almak istediğinde IslevKullanıcısı içindeki ilgili fonksiyonu çağırır.



Şekil 8: İşlev Takipçi Yapısı

IslevKullanicisi icindeki fonksiyonlar "virtual" olarak tanımlanmıştır. Bu işlev yapısını kullanmak isteyen YKB "IslevTakipcisi" sınıfında fonksiyonları gerçekler.

İşlev Takipçi Yapısı için örnek sınıf diyagramı Şekil 8'de verilmiştir.

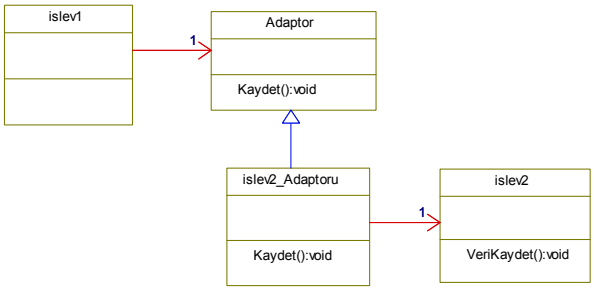
Yukarıda anlatılan yapı bir sorumluluğu yerine getiren işlev yapısının çevresindeki diğer işlevlerle olan ilişkilerini IslevKullanicisi içinde toplar. Bu sayede işlev yapısı tek ve bütün bir yapı olarak Referans Modelde kullanılabilir. İşlev yapısını kullanmak isteyen YKB bu yapıyı modeline ekler ve IslevTakipçisi sınıfını gerçekler.

5.5. Adaptör Yapısı

Referans Model oluşturulurken Adaptör yapısının faydaları açıktır. Kullanılmış ve test edilmiş yapıları modelde kullanmak istediğimizde uyumsuz arayüzler kaçınılmaz olarak ortaya çıkar. Bu arayüzlerin birbiriyle uyumlandırılması için Adaptör Yapısı kullanılır.

Adaptör yapısı bir sınıfın arayüzünü bu sınıfı kullanacak diğer sınıfın istediği arayüze çevirme işlemidir. Adaptör yapısı uyumsuz arayüzlere sahip sınıfların birlikte çalışabilmeleri için bir yapı sunar[5].

Örnek kullanımı Şekil 9'daki sınıf diyagramında gösterilmiştir.



Şekil 9: Adaptör Yapısı

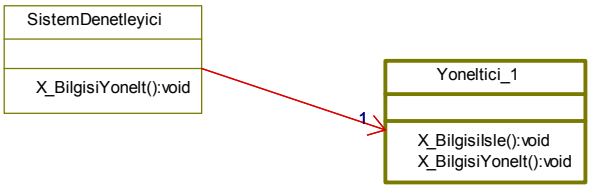
Bu yapıda işlev1 sınıfı, Adaptör sınıfının sağladığı bir arayüz olan "Kaydet()" fonksiyonunu kullanır. Bu arayüz Islev2_Adaptörü içindeki fonksiyon içinde Islev2'nin sahip olduğu "VeriKaydet()" fonksiyonu kullanılarak gerçeklenir.

5.6. Sistem Denetleyici Yapısı

Sistem Denetleyici Gömülü Kontrol Yazılımlarında önemli bir görevde yer alır. Sistem Soyutlama katmanından gelen mesajlar bu katmanda ilgili yerlere yönlendirilirler.

Sistemin görevleri arttıkça Sistem Denetleyici'nin görevleri de artar. Aktif kontroller yapmak ihtiyacı ve takip edilebilirliğin zorlaşması sonucunda SistemDenetleyici sınıfının görevlerinin azaltılması gerekebilir. Bu amaçla yeni bir yapı oluşturulabilir. Bu yapıda Sistem Soyutlama seviyesinden gelen mesajlarla ilgili görevlerin bir kısmı SistemDenetleyicide bir kısmı da oluşturulan yeni sınıfta yer alabilir.

Yeni yapı şu şekilde gerçekleştirilir; bir mesajın kullanılabileceği farklı görevler varsa bu mesajın içeriği öncelikle SistemDenetleyici'de ayrıştırılır bazı görevler yerine getirilir, aynı mesaj diğer sınıfa aktarılır ve kalan görevleri de bu sınıf yerine getirir.



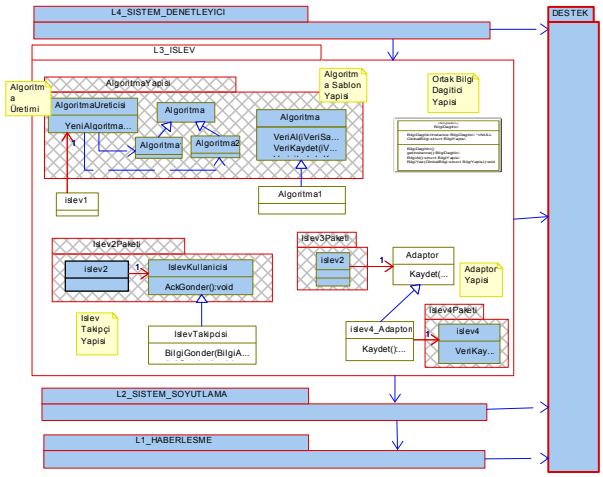
Şekil 10: Sistem Denetleyici Yapısı

Sistem Denetleyici Yapısı için örnek sınıf diyagramı Şekil 10'da verilmiştir.

Bu yapıda dikkat edilmesi gereken önemli nokta Sistem Soyutlama Katmanı'ndan gelen mesajın birden fazla ve farklı görev için kullanılabilir olmasıdır. Sistem Soyutlama seviyesi'nden alınan mesaj Sistem Denetleyici ve Yöneltici nesneleri içinde ayıklanarak işlev gerçekleştirilir.

6. İşlev Katmanı Referans Modeli

İşlev Katmanında kullanılan Referans Model için örnek gösterim Şekil 11'de verilmiştir.



Şekil 11: Referans Model Örnek Yapısı

Bölüm 2'de verilen Katmanlı Mimari Yapının, İşlev Katmanı'nda kullanılan yöntemlerle birleştirildiği hali Şekil 11'de gösterilmiştir. Bu gösterimde Mimari yapıdaki İşlev Katmanı paketler, sınıflar ve aralarındaki ilişkilerle detaylandırılmıştır.

Şekil 11'de İşlev Katmanı içindeki mavi renkle boyalı yapılar referans modelden kullanılabilir yapıları ifade etmektedir, beyaz renkle gösterilen yapılar ise uygulamaya özel geliştirilebilir yapılardır.

7. Kazanımlar

Bu çalışma sonucunda Gömülü Kontrol Yazılımları İşlev Katmanı'nda yeniden kullanılabilir yapılar oluşturulmuş ve bu yapılar Referans bir Modele aktarılmıştır. Farklı görevleri olan 2 YKB bu referans modeldeki ortak yapıları kullanacak şekilde yeniden yapılandırılmış ve başarılı olunmuştur.

Test edilmiş ortak yapılar birden fazla YKB tarafından kullanılmakta ve bu yapılara olan güven artmaktadır. Ortak yapı ile ilgili bir hata tespit edildiğinde düzeltme Referans modelde yapılmakta dolayısıyla artıklıktan kurtulunmakta ve bu düzeltme 2 YKB'ye de yansımaktadır.

Referans Modeldeki bir yapıyı kullanmak isteyen kişi hazır test edilmiş bu yapıyı alabilmekte ve küçük bir işçilikle bu yapıyı YKB'ye entegre edebilmektedir. Bu şekilde benzer işlevler için 2 YKB'de ayrı ayrı harcanabilecek fazla işçiliğin önüne geçilmektedir.

YKB Gerek gerçeklenme sayısı ve zamanı gösteren tablo Tablo 1'de verilmiştir. Bu tablo kodlama ve tasarımı yeni başlamış bir YKB'nin gereklerinin gerçeklenme durumlarını gösteren bir tablodur. Tablodan görüldüğü gibi YKB'nin tasarımı Kasım ayında başlamıştır. Kasım, Aralık ve Ocak aylarında kodlanan gerek sayısı olağan bir artışla devam ederken Ocak ve Şubat aylarındaki kodlanan gerek sayısı önemli sayıda artış göstermiştir. Bunun sebebi bu zaman diliminde Referans Model kullanımına ve ortak kullanılan işlev katmanı yapılarına geçilmesidir.

Tablo 1: Gerek Gerçeklenme/Süre Tablosu

Tarih	YKB Gereksinim sayısı	Kodlanan Gerek Sayısı
Kasım 2008	111	9
Aralık 2008	111	15
Ocak 2009	111	**38**
Şubat 2009	122	**70**
Mart 2009	127	75

Tablo 2'de referans model kullanımı sürecinde YKB boyutunun ve karmaşıklık değerlerinin zamanla değişimi gösterilmiştir. Tabloda görüldüğü gibi Referans Model'e geçişle birlikte yeni işlevler eklenmiş fakat ortalama karmaşıklık değeri düşmüştür. Karmaşıklığın artmak yerine azalması Referans Model kullanımının kazanımlarından biri olmuştur.

Tablo 2: Karmaşıklık ve Sınıf Sayıları/Süre Tablosu

Tarih	YKB Boyutu (LOC)	Sınıf Sayısı	Ortalama Çevrimsel Karmaşıklık Değeri
Kasım 2008	4435	45	1.61
Aralık 2008	9843	81	1.8
Ocak 2009	17088	112	1.78
Şubat 2009	22340	152	1.6
Mart 2009	22435	155	1.6

Referans Model Kullanımı için kodun yeniden yapılandırılması ile tasarım daha anlaşılabilir olmuştur. Anlaşılabilirliğin artmasında kullanılan tasarım yöntemlerinin önemi açıktır.

Ortak bir tasarım oluşturulması tasarımı bilen ve gözden geçiren kişi sayısını artırmaktadır.

Tasarımda kullanılan yöntemler işlev yapıları arasında bağımsızlığı sağlamıştır. Bu yüzden bir diğer işlevi tasarıma eklemeden, ilgili yapı içinde değişiklik yapmak kolaylaşmıştır.

İşlev katmanı'nda yapılar arasındaki bağımlılıklar azaldığı için tasarımın idame ettirilebilirliği artmıştır.

8. Karşılaşılan Zorluklar

Bu çalışma ile önemli kazanımlar elde edilmiştir. Bununla birlikte yeniden yapılandırma sürecinde karşılaşılan bazı zorluklar da olmuştur. Bu zorluklar 2 başlıkta toplanabilir:

Proje Takvimi: Referans Model oluşturulması için yeniden yapılandırma, devam eden bir projede gerçekleştirildiğinde Proje Takvimi'nin oluşturduğu bazı baskılar oluşmaktadır.

- Başlanan bir iş tamamlanmak zorundadır. Yarım bırakma, daha sonra bu işe geri dönme gibi seçenekler bulunmamaktadır.
- Geliştiriciler bu işe odaklanmak durumunda oldukları için yeni işlere servis veremezler. Bu da Proje yöneticilerinden yeni bir işlev eklenmesi isteği geldiğinde onlara servis verememe anlamına gelir ki bazı durumlarda bu probleme yol açabilir.

Yazılım Dallanma'ları: Yeniden Yapılandırma çalışmaları ile yazılımda oluşturulan dallanmaların bir araya getirilmesinde zorluklarla karşılaşılabilmektedir. Bu yüzden konfigürasyon takibi yapılırken özellikle dikkat edilmesi gerekmektedir.

9. Sonuçlar

Bu bildiride Gömülü Kontrol yazılımlarında İşlev Katmanı için Referans Modeli oluşturmak amacıyla yapılan çalışmalar anlatılmıştır. Mevcut Kullanılan Referans Yazılım Mimarisi verilmiş ve İşlev Katmanı detaylı olarak incelenmiştir.

İşlev Katmanı'nda Referans Model oluşturmak için kullanılan yöntemler detaylı olarak açıklanmış ve sunulmuştur.

Elde edilen kazanımlar oldukça önemlidir. Bu yüzden bu çalışmanın ileriye dönük tartışmalar için bir başlangıç olarak düşünülebileceği ve devam ettirileceği değerlendirilmektedir.

10. Teşekkür

Bu makalede anlatılan tasarımın oluşturulmasına destek veren ve değerli fikirlerini paylaşan tüm çalışma arkadaşlarıma teşekkür ederim.

11. Kaynakça

[1] Gören, H. Ö., Gürler, E., "Elektronik Harp Sistemleri Gömülü Kontrol Yazılımı Mimarisi", *UYMK 2006*

[2] Medvidovic N., Malek S., Rakic M. M., "Software Architectures and Embedded Systems", Proceedings of the Monterey Workshop on Software Engineering for Embedded Systems (SEES 2003), pages 65-71, Chicago IL, September 24-26, 2003

[3] Buschmann F., Meunier R., Rohnert H., Sornmerlad P., Stal M., "Pattern Oriented Software Architecture:A System of Patterns", Volume–1, John Wiley&Sons, 1996

[4] C. Larman, "Applying UML and Patterns:An Introduction to Object Oriented Analysis and Design and the Unified Process", 2/e, Prentice Hall, 2002

[5] Gamma E., Helm R., Johnson R., and Vlissidies J., "Design Patterns - Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.