Gömülü Sistemlerde Model-View-Controller (MVC) Kullanımı

İbrahim Sıcakyüz, Metin Tekkalmaz, Murat Salman

REHİS-EHGYM, Aselsan A.Ş. , Ankara
{sicakyuz,tkalmaz,msalman}@aselsan.com.tr

Özet. Model-View-Controller (MVC) tasarım kalıbı, verinin farklı görselleştirme yöntemleriyle kullanıcıya sunulduğu uygulamaların geliştirilmesinde kullanılan bir yöntemdir. Bu çalışmada, genellikle uygulama yazılımlarında ve web uygulamalarında kullanılan MVC tasarım kalıbının gömülü sistemlere uyarlanmasından bahsedilmektedir. İlk olarak gömülü sistemlerde MVC tasarım kalıbının uygulanması açısından farklılıklar verilmekte, ardından farklı görüntü birimlerine sahip bir projede MVC tasarım kalıbının katmanlı mimari içerisinde nasıl kullanıldığı aktarılmakta, son olarak da bu yaklaşımın farklı görüntü birimleri ve kullanıcı girdi arayüzleri bulunan gömülü sistemlerde uygulanmasına yönelik genel bir yaklaşım önerisi sunulmaktadır.

Anahtar Kelimeler: model, view, controller, gömülü yazılım, mimari, kullanıcı arayüzü

Abstract. Model-View-Controller (MVC) design pattern is a method that is used in development of applications those data is presented to the user by different methods. In this work, an approach to adapt MVC design pattern, which is usually used in application software and web applications, to use in embedded systems is described. First, the differences of embedded systems considering MVC design pattern are given. Then, the usage of MVC design patterns in a system that has different display units and employs a layered architecture in software design is presented. Lastly, a general approach to use MVC design pattern in development of embedded systems with different presentation devices and user input interfaces is proposed.

Keywords: model, view, controller, embedded software, architecture, user interface

1 Giriş

Uygulama yazılımları basit bir yaklaşımla veri ve bu verinin değiştirilmesine yönelik kullanıcı etkileşimi sağlayan kullanıcı arayüzü kısımlarından oluşur. Kullanıcıya sunulacak veri ile bu verinin kullanıcıya nasıl sunulacağı temelde

birbirinden farklı kavramlardır. İlk olarak Smalltalk-80 programlama dilinde, kullanıcı arayüzü geliştirmesi için ortaya konan çerçevede (İng. framework) kullanılan Model-View-Controller (MVC) yaklaşımı [1], verinin saklanması ve kullanıcı etkileşimi kavramlarını birbirinden bağımsız tutmayı amaçlayan bir mimari tasarım kalıbı olarak kabul görmüştür. En basit anlamda Model verinin saklanmasından, View verinin kullanıcıya sunulmasından, Controller ise kullanıcı girdilerine göre verinin değiştirilmesinden sorumludur.

Gömülü sistemler için geliştirilen yazılımlar, uygulama yazılımlarına göre birçok farklılık göstermektedirler. Bu iki tür yazılım kullanıcı etkileşimi açısından karşılaştırıldığında şu fark ön plana çıkmaktadır: Uygulama yazılımları genel amaçlı bilgisayarlar üzerinde koşmak üzere tasarlanmışlardır, dolayısı ile kullanıcı girdi (klavye, fare v.b.) ve çıktıları (ekranlar, grafiksel kullanıcı arayüzü altyapıları v.b.) açısından standart altyapılar ile etkileşirler. Gömülü sistemler ise belli bir amaca yönelik tasarlanmışlardır ve özelleşmiş donanım parçalarından oluşurlar. Bununla paralel olarak gömülü sistemler kullanıcı etkileşimi açısından çok farklı yeteneklere sahip olabilmektedir. Bazı sistemlerin kullanıcı arayüzü bulunmamakta, bazılarının lamba/LED gibi çok basit, bazılarının ise daha karmaşık ekranları bulunabilmektedir. Gömülü sistemlerin bu özelliği nedeni ile bu sistemler üzerinde koşan yazılımların da çok farklı altyapılar ile çalışma gereksinimi bulunmaktadır.

Bu makalede, temelde klasik uygulama yazılımları için ortaya konulmuş, zamanla web uygulamalarında da yaygın kabul görmüş, gömülü sistem olarak küçük ölçekli Linux uygulamalarında [2] [3] kullanılmış olan MVC tasarım kalıbının gömülü sistemler için geliştirilmiş yazılımlarda nasıl kullanılabileceğine dair bir öneri sunulmaktadır.

Makalenin içeriği şu şekildedir: Bölüm 2'de farklı çeşitteki hava platformlarında çalışmak üzere tasarlanan gömülü bir sistem MVC tasarım kalıbı kullanımı açısından tanıtılmaktadır. Bölüm 3'te ise Bölüm 2'de anlatılan sistemde kazanılan tecrübeler ve daha geniş bir bakış açısı ile gömülü sistemlere yönelik MVC kullanım önerisi verilmektedir. Son olarak Bölüm 4'te değerlendirmeler ile makale tamamlanmaktadır.

2 MVC Tasarım Kalıbının Projede Kullanımı

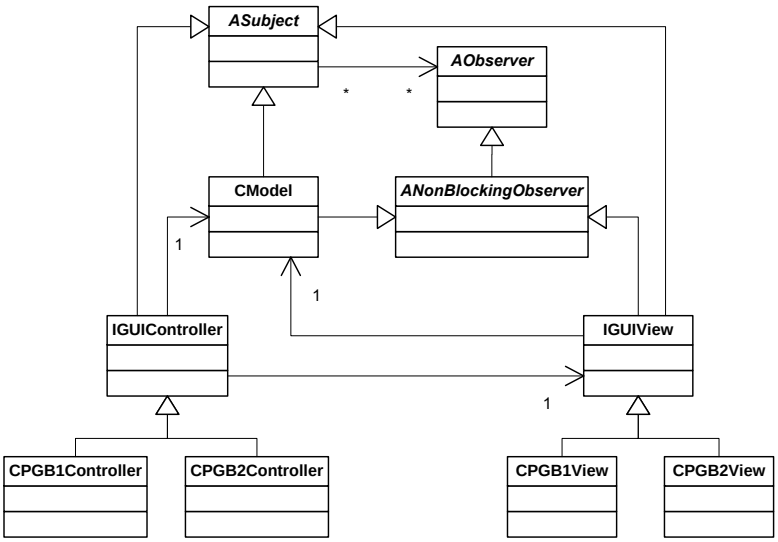
Aselsan bünyesinde geliştirilmiş olan bir kendini koruma sisteminin monte edildiği farklı hava platformlarında iki çeşit görüntü biriminin kullanılması ihtiyacı ortaya çıkmıştır. Platformların genelinde PGB-1 (Pilot Görüntü Birimi - 1) çok fonksiyonlu görüntü biriminin kullanılmasına, yerleşim kısıtlarından dolayı bu görüntü biriminin kullanılamadığı platformlarda ise PGB-2 görüntü biriminin kullanılmasına karar verilmiştir.

Proje kapsamında geliştirilmiş olan Yönetim YKB'nin (Yazılım Konfigürasyon Birimi), kullanıcı arayüzü olarak birbirinden oldukça farklı olan bu iki görüntü birimini de kullanabilmesi gereğinin gerçeklenmesi için yazılım geliştirme aşamasında Model-View-Controller (MVC) tasarım kalıbının kullanılması değerlendirilmiştir. Bu iki görüntü biriminin sunacağı veri ve bu verinin temin edileceği altsistemler aynı olsa da görüntü birimlerinin kullanılması aşamasındaki

farklılık hem görüntü birimine gönderilecek verilerin gönderim şeklinden hem de görüntü biriminden alınacak verilerin iletim şeklinden kaynaklanmaktadır. Bu farklılıkların üstesinden gelmek için Yönetim YKB kapsamında MVC tasarım kalıbının kullanılmasına karar verilmiştir. Buna göre kullanıcıya sunulacak veri bir Model sınıfında tutulmuş, kullanıcı arayüzü ile etkileşimin gönderim aşamasında View sınıfları, alım aşamasında ise Controller sınıfları kullanılmıştır.

Yapılan tasarım çalışmaları neticesinde katmanlı bir mimariye sahip olacak şekilde tasarlanan Yönetim YKB, 4 katmandan oluşan bir yapıda geliştirilmiştir. Bu katmanların en alt seviyesinde altsistemler ile haberleşmede kullanılan protokolleri içeren "Ham Veri Haberleşme (L1)" katmanı yer almaktadır. İkinci seviyede, altsistemler ile haberleşme ve altsistemlerin kontrol işlemlerinin gerçekleştirildiği "Altsistem Haberleşme (L2)" katmanı, üçüncü seviyede ise sistemin çalışması sırasında gerçekleştirdiği aktivitelerin kontrollerinin sağlandığı "İşlev (L3)" katmanı bulunmaktadır. Dördüncü ve en üst katman olarak da sistemin kullanıcı ile arayüzünü sağlayan görsel ve işitsel aktivitelerin kontrol edildiği "Sunum (L4)" katmanı belirlenmiştir. Katmanlar arasındaki ilişkilerin düzenlenmesi aşamasında ise üst katmanda yer alan sınıfların alt katmanda yer alan sınıflar ile doğrudan ilişki kurabilmesi kararlaştırılırken alt katmanda yer alan sınıfların üst katmanda yer alan sınıflara bağımlı olmaması amacıyla Observer tasarım kalıbı [4] kullanılmasına karar verilmiştir.

Yönetim YKB kapsamında kullanıcı arayüzünde sunulacak olan tüm veriler L2 katmanında yer alan altsistem kontrol sınıflarından ve L3 katmanındaki işlev sınıflarından üretilen bilgiler olduğundan Model sınıfının tüm bu verileri alabilecek bir konumda yer alan L4 katmanı içerisinde konumlandırılmıştır. L4 katmanında bulunan Model sınıfının sistemin çalışması ile ilgili bilgileri bu bilgileri üreten sınıflardan alabilmesi için yazılım genelinde kullanılan Observer tasarım kalıbı kullanılmıştır.



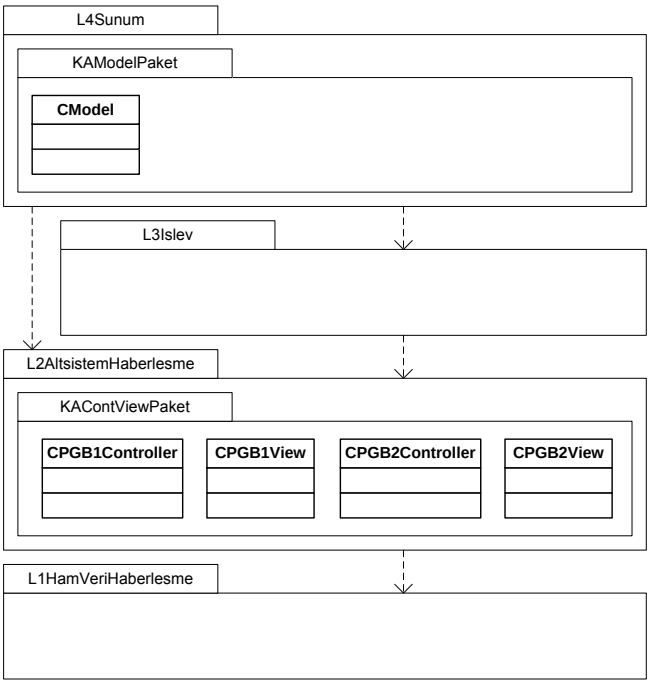
Şekil 1. MVC Sınıfları ve Birbirleri ile İlişkileri

MVC tasarım kalıbının Yönetim YKB tasarımında kullanılmasına sebep olan farklı görüntü birimleri ile etkileşim gereksinimi neticesinde her iki görüntü biriminin denetlenmesi ve kontrol edilmesi için her bir görüntü birimi için birer adet olacak şekilde View ve Controller sınıfları geliştirilmiştir. Katmanlı mimari ile tasarlanmış uygulama yazılımlarında View ve Controller sınıfları da genel olarak sunum katmanında yer almaktadır. Ancak projede görünüm ve kullanıcı girdisi sağlayan kısımlar yazılımın bir parçası değil ayrı birimler olduğundan View ve Controller sınıfları aynı zamanda birer altsistem haberleşme sınıfı olarak tasarlanarak L2 katmanında yer almışlardır. MVC tasarım kalıbı ile doğrudan ilgili sınıflar Şekil 1'deki UML çizelgesinde verilmiştir.

MVC tasarım kalıbının genel kullanım şekilleri içerisinde kullanıcı girdilerinin takip edilmesi ve bunların neticesinde gerçekleştirilmek istenen işlevlerin belirlenmesi için iki farklı yaklaşım söz konusudur. Birinci yaklaşımda kullanıcı tarafından yapılan bezel tuşu basımları gibi hareketler View tarafından algılanıp bu hareketler neticesinde yapılacak olan işlem belirlenerek gerekli yerlere iletilmesi için Controller sınıfına aktarılır. Diğer yaklaşımda ise kullanıcı tarafından yapılan hareketler direk olarak Controller sınıfından takip edilerek bu hareketler neticesinde gerçekleştirilecek işlemler kontrol edilir. Görüntü birimi olarak kullanılan PGB-1 birimi kullanıcının yaptığı işlemleri kendi yazılımı dâhilinde değerlendirerek Yönetim YKB'ye sadece bu işlemler neticesinde yapılmak istenen işlemi ya da değiştirilmek istenen seçimi bildiren bir arayüze sahip akıllı bir birimdir. Bu nedenle tasarım aşamasında bu birim için ikinci uygulama olarak bahsedilen direk olarak Controller sınıfı üzerinden kullanıcı isteklerinin takip edilmesi yöntemi belirlenmiştir. Controller sınıfı tarafından alınan kullanıcı hareketleri doğrultusunda güncellenecek olan verilerin, Model sınıfına iletilmesi aşamasında ise MVC tasarım kalıbının yapısının Yönetim YKB için benimsenmiş olan katmanlı mimari yapısı ve Observer tasarım

kalıbı yapısına uymamasından ötürü bazı uyarlamalar yapılması zorunlu hale gelmiştir. L2 katmanındaki Controller sınıfının L4 katmanındaki Model sınıfı ile ilişki kurmasına ve L4 katmanındaki Model sınıfının L2 katmanındaki Controller sınıfı ile Observer yapısı kurması şeklinde yapılan değişiklik ile mimari tasarım MVC tasarım kalıbına uygun hale gelecek şekilde değiştirilmiştir. Benzer bir durum Controller sınıfı tarafından alınan kullanıcı isteklerinin L3 ve L2 katmanındaki diğer sınıflara iletilmesi için Controller sınıfı ile L3 katmanındaki ve L2 katmanındaki diğer sınıflar ile direk ilişkisi olması gerekli iken katmanlı mimari kararları gereğince bu tür ilişkilerin kurulmasına müsaade edilmemesi nedeniyle de yaşanmıştır. Bu durumun çözümünde ise Controller ve Model sınıfları arasındaki ilişki yönü sorunun çözümü için uygulanan değişiklikten faydalanılmıştır. MVC yapısında kullanıcıdan gelen komutlar neticesinde yapılması gereken işlevlerin ilgili sınıflara iletiminin Controller sınıfı tarafından yapılması prensibi mevcuttur. Bu prensipten bir miktar değişikliğe gidilerek bu komutların öncelikli olarak Controller sınıfının direk ilişkisi olan Model sınıfına iletilmesi, L3 ve L2 katmanındaki sınıflara iletimin ise L4 katmanında yer alan ve L3 ve L2 katmanında yer alan sınıflar ile direk ilişkisi bulunan Model sınıfı üzerinden gerçekleştirilmesi için uyarlamalar yapılmıştır.

PGB-2 için eklenen Controller sınıfında da daha önce bahsedilmiş olan iki çeşit kullanıcı hareketi ve isteği takip etme yönteminden yine kullanıcı hareketlerinin direk olarak Controller sınıfı tarafından alınması prensibine dayanan ikinci yöntem benimsenmiştir. Ancak PGB-2 biriminin PGB-1'den farklı olarak sadece tek tek üzerinde yer alan bezel tuşları ile yapılmış olan basım bilgilerini göndermesinden ötürü kullanıcı tarafından yapılan işlemlerin belirlenmesinde PGB-1 birimi için uygulanan yöntemden farklı bir yaklaşım sergilenmiştir. PGB-2 için yapılan tasarımda bezel tuşu basımlarının yapılmış olduğu aktif olan sayfanın hangi sayfa olduğu bilgisine ve aktif sayfanın sayfa yerleşimi bilgisine sahip olan sınıf View sınıfıdır. Haberleşme arayüzü üzerinden gelen ve PGB-2 için oluşturulmuş olan Controller sınıfı tarafından alınan bezel tuşu basımı bilgilerinin neticesinde yapılmak istenen işlemi tespit edecek şekilde yorumlanması için View sınıfına aktarılmıştır. Controller sınıfından View sınıfına aktarılan bezel tuşu basım bilgileri View sınıfı içerisinde değerlendirilerek elde edilen kullanıcı istekleri ya da veri değişiklikleri PGB-1 birimi için tariflenen Controller-Model arayüzüne benzer şekilde View-Model arayüzü ile Model sınıfına aktarılacak şekilde tasarlanmıştır. View ve Model sınıfları arasındaki bu ilişkinin kurulabilmesi için ise PGB-1 biriminin Controller ve Model sınıfları arasındaki MVC tasarım kalıbına uygun olarak tanımlanan ilişkinin katmanlı mimari tasarımına uygun olmaması şeklinde bir sorun PGB-2 birimi için de View sınıfı ile Model sınıfı arasında ortaya çıkmıştır. Bu zıtlık için PGB-1'de olduğu gibi katmanlı mimarinin tasarım kararlarından bir miktar feragat edilerek L2 katmanında yer alan View sınıfının L4 katmanında yer alan Model sınıfı ile ilişki kurması, bunun karşılığında Model sınıfının ise View sınıfı ile olan ilişkisini Observer yapısı üzerinden gerçekleştirmesi için gerekli değişikler yapılmıştır.



Şekil 2. Katmanlı Mimari İçerisinde MVC Sınıfları

MVC tasarım kalıbının son parçası olan View sınıflarının eklenmesinde View sınıfında kullanılacak olan verilerin güncel değerlerinin Model sınıfından alınması için katmanlı mimari yapısının tasarım kararlarına uymamakla birlikte daha önce yapmış olduğumuz düzenlemeler neticesinde tasarım kararlarını MVC tasarım kalıbının kurallarına uygun hale getirecek şekilde değiştirerek oluşturduğumuz Observer yapısı kullanılmıştır. Model sınıfının L4 katmanına yerleştirilmesi sırasında sistemin çalışması hakkında kullanıcıya sunulacak olan bilgileri üretmekle sorumlu olan altsistem ve işlev denetleyici sınıflarına Observer yapısını kullanarak abone olması sağlanmıştır. Model, View ve Controller sınıflarının Yönetim YKB'nin katmanlı mimarisi içerisine yerleştirilmesi sonucunda Şekil 2'deki yapı elde edilmiştir.

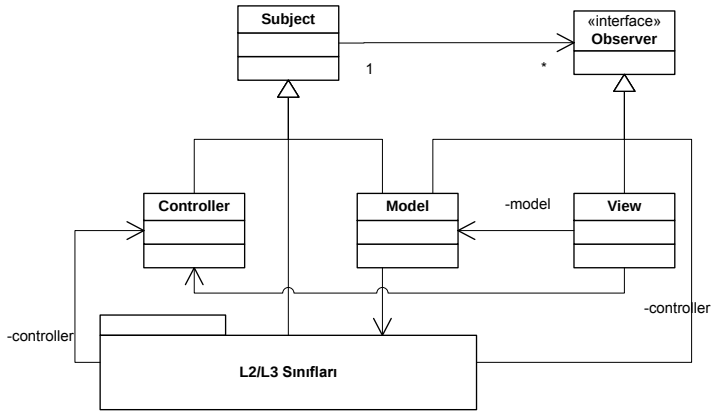
3 Gömülü Sistemler için MVC Önerisi

MVC tasarım kalıbı genel olarak masaüstü uygulamaları için ortaya atılmış ve zamanla web uygulamalarında da kabul görmüş bir yaklaşım olsa da bu yaklaşımın Bölüm 2'de anlatıldığı gibi gömülü sistemlerde de kullanımı mümkündür. Bölüm 2'de aktarılan yaklaşım farklı platformlar üzerinde farklı görüntü birimleri ile çalışabilen bir yazılım için tasarlanmıştır. Bu yaklaşım her ne kadar farklı görüntü birimleri (dolayısı ile faklı sunum yöntemleri) kullanımını yazılımın geri kalandan

soyutlamış olsa da tasarım gereksinimleri (kullanıcıya sunulacak bilgiler, kullanıcıdan alınacak girdiler, birlikte çalışılacak görüntü birimleri v.b.) belli bir proje için ortaya konmuştur. Dolayısı ile ortaya konulan genel yaklaşımı farklı projelere yansıtmak mümkün olmakla birlikte ayrıntılı tasarımı ya da ilintili gerçeklemeyi farklı projelerde kolayca yeniden kullanmak mümkün gözükmemektedir.

Bu bölümde farklı gömülü yazılımlarda detay tasarım ve ilintili gerçekleme seviyesinde yeniden kullanımı mümkün kılacak MVC tasarım kalıbına dayanan bir kullanıcı etkileşim altyapısı önerisi verilmektedir. Öneri, yeniden kullanımda hedef gömülü yazılımların belli bir alana (örn. kendini koruma süit yönetimi) yönelik geliştirildiğini varsaymaktadır. Aksi durumda Model ve Controller'ın ortaklanması ihtimali düşmektedir. Örneğin bir kendini koruma süit projesi kapsamında kullanıcıya tehdit (füze, radar, lazer v.b.), karşı tedbir, cihaz içi test sonucu, altsistem durumu, sürüm gibi bilgilerin sunulacağı; kullanıcıdan da mod seçimi, karşı tedbir onayı, test başlatma isteği gibi girdilerin alınacağı bellidir. Bu bilgi Model, View ve Controller arası arayüzlerin belirlenebilmesini mümkün kılmaktadır.

Önerinin diğer bir hedefi, klasik anlamda değerlendirildiğinde ilk bakışta kullanıcı arayüzü olarak değerlendirilmeyecek diğer kullanıcı etkileşim yöntemlerini de MVC yapısına dâhil etmektir. Bu etkileşim yöntemleri arasında LED/lamba ile bilgi sunumu, kontrol paneli butonları üzerinden kullanıcı girdisi, sesli uyarılar gibi yöntemler bulunmaktadır.

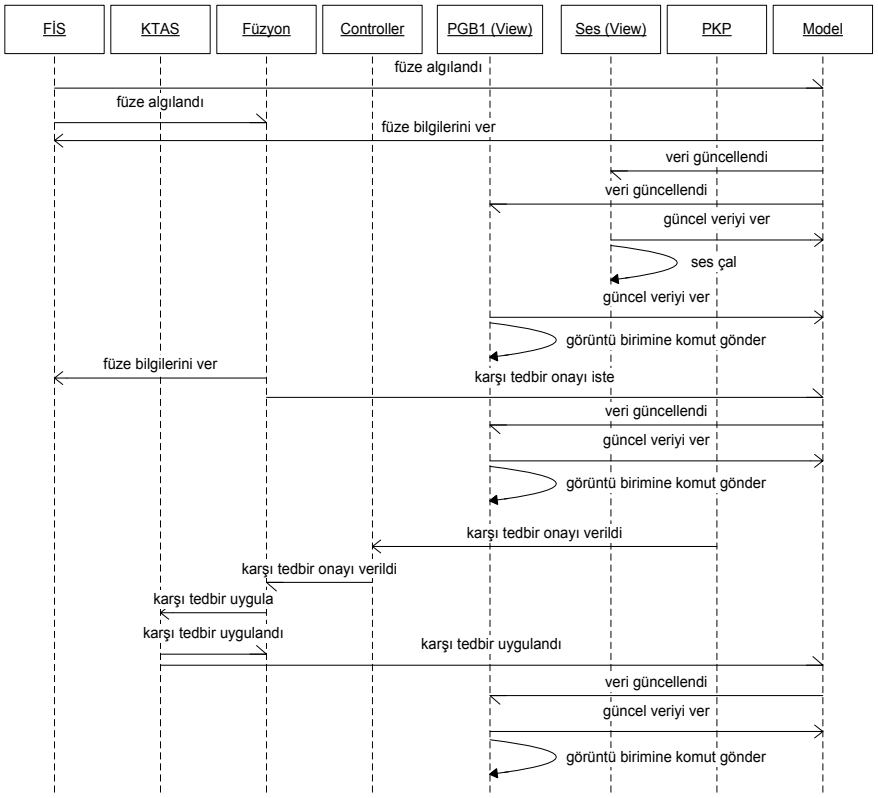


Şekil 3. Gömülü Sistemler için MVC Önerisi

Gömülü sistemler için önerilen MVC yapısı Şekil 3'de bir UML çizelgesinde verilmiştir. Model-View-Controller sınıflarının kendi aralarındaki ilişkiler, Model-View-Controller sınıflarının yazılımdaki diğer sınıflar ile ilişkileri ile Subject/Observer rolleri çizelgede sunulmuştur. Model klasik MVC yaklaşımına benzer şekilde kullanıcıya sunulacak tüm veriyi saklamakla yükümlüdür ancak yazılımın asıl durumu L2 ve L3 katmanlarında yer alan sınıflarında yer almaktadır. L4 katmanında yer alan Model sınıfı L2/L3 sınıflarından ihtiyaç duyduğu veriyi Observer tasarım kalıbı çerçevesinde elde etmektedir. Önerinin hedef aldığı gömülü sistemlerde yazılımda durum değişikliğine neden olan girdiler kullanıcıdan olduğu kadar yazılımın haberleştiği algılayıcı (İng. sensor) ve eyleyici (İng. actuator) birimler

aracılığı ile de gerçekleştiğinden, View sınıflarının Controller tarafından tetiklendiği pasif MVC yaklaşımı yerine Model tarafından tetiklendiği aktif MVC yaklaşımı tercih edilmiştir. Dolayısı ile Model sınıfının tuttuğu veri güncellendiğinde bu durum yine Observer tasarım kalıbı ile duyurulmakta, durum değişikliğinden haberdar olan View sınıfları da tiplerine göre görüntü birimi ile gerekli haberleşmeyi yürütme, ayrık hatlar üzerinden lambanın durumunu değiştirme, ses modülü üzerinden uygun sesi üretme gibi işlemleri yürütmektedirler. Buraya kadar anlatılan yöntem View kavramının genişletilmesi dışında Bölüm 2'de anlatılan projede uygulanan tasarım ile uyumludur.

Önerilen MVC yapısındaki projede uygulanan tasarıma ve klasik MVC yaklaşımına göre en belirgin fark Controller sınıfı ile ilintilidir. Controller sınıfı, kullanıcı girdilerinin belli bir alan (İng. domain) için ortaklandığı ve Mediator tasarım kalıbındaki [4] Mediator sınıfına benzer işleve sahip bir sınıf olarak tasarlanmıştır. Çok farklı kullanıcı arayüzleri (akıllı/akılsız görüntü birimlerine ait bezel tuşları, kontrol panelleri üzerindeki düğmeler v.b.) üzerinden alınan kullanıcı girdilerine göre Controller sınıfının ilgili metodu çağrılmaktadır. Önerilen yapıda kullanıcı girdilerini anlamlandıran sınıflar View sınıfları ya da (kontrol paneli örneğinde olduğu gibi) bir View sınıfı olmasa da ilgili kullanıcı girdisini dinleyen sınıflardır. Controller sınıfı kendisine bildirilen kullanıcı girdisini Observer tasarım kalıbı çerçevesinde duyurmakta, Controller sınıfı için Observer rolünde bulunan L2/L3 sınıfları da duyurulan kullanıcı girdisine göre durumlarını değiştirmektedir. Önceki paragrafta anlatıldığı gibi L2/L3 sınıflarındaki değişimden haberdar olan Model sınıfı View sınıflarını tetikleyerek kullanıcı arayüzünün güncellenmesini sağlamakta böylece Model-View-Controller döngüsü tamamlanmaktadır. Bu yaklaşımda Controller sınıfına düşen işlev oldukça sınırlı gözükmektedir. Ancak farklı projelere göre farklılık gösterebilecek View sınıfları ve belli bir proje için birden fazla sayıda yer alabilecek olası Controller sınıfları ile L2/L3 sınıfları arasındaki bağımlılığı kırmak açısından önerilen yaklaşımdaki Controller sınıfı önem taşımaktadır.



Şekil 4. Önerilen MVC Yapısı Üzerinde Örnek Bir Akış

Şekil 4'de, gömülü sistemler için önerilen MVC yaklaşımının bir kendini koruma süit projesi kapsamında uygulanması durumunda oluşabilecek örnek bir akış mesaj sıra çizelgesi (İng. message sequence chart) olarak verilmiştir. Çizelgede metot çağrımları çift kanatlı ok işareti, Observer yapısı ile bildirimler tek kanatlı ok işareti ile gösterilmiştir. Observer yapısı ile bildirimler birden fazla Observer'a gidiyor olsa da akışta yalnızca anlam taşıyan bildirimler gösterilmiştir. Örnek akış füze ikaz sistemi (FİS) ile iletişimi sağlayan sınıftan tehdit bildirimi ile başlamaktadır. Bu bildirimle Füzyon karşı tedbir kararı ile Model ise tehdidin kullanıcıya sunumu için ilgilenmektedir. Tehdit MVC yapısı içinde kullanıcıya görsel ve sesli olarak sunulmaktadır. Füzyon ise karşı tedbire karar vererek kullanıcıdan onay gerekliliğini bildirmektedir. Onay isteği yine MVC yapısı ile kullanıcıya görüntü birim üzerinden sunulmaktadır. Ancak örnek akış içerisinde onay görüntü birimi üzerinden değil onay işlevini sağlayan pilot kontrol paneli (PKP) üzerinden verilmektedir. Kullanıcı girdisi Controller yapısı üzerinden ilgili sınıflara bildirilmekte ve akış karşı tedbirin

uygulanması ve karşı tedbirin uygulandığı bilgisinin MVC yapısı üzerinden kullanıcıya sunulması ile sonlanmaktadır.

4 Sonuç

Model-View-Controller (MVC), kullanıcıya sunulacak verilerin farklı görselleştirme yöntemleri ile hazırlanmış kullanıcı arayüzlerinden soyutlanarak saklanması ve güncellenmesi için geliştirilmiş bir tasarım kalıbıdır. Makalede MVC tasarım kalıbının bir gömülü sistem yazılımı kapsamında neden ve nasıl kullanıldığı verilmiştir. Ardından da bu kullanımdan elde edilen tecrübelerle MVC tasarım kalıbının katmanlı mimari uygulanan gömülü sistem yazılımlarında uygulanmasına yönelik genel bir yaklaşım önerilmiştir. Önerilen yöntemin yeni projeler kapsamında uygulanmasına başlanmıştır.

Kaynaklar

1. Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, 1(3):26–49, August/September 1988.
2. Jivani P., Chopara C., Prashant M., Over All Idea about MVC: How to use Model-View-Controller (MVC), International Journal of Innovations in Engineering and Technology (IJIET), 391-395, Vol. 2 Issue 1, February 2013.
3. Bajpai A., Model View Controller Architecture on Embedded Systems, DAE-BRNS Seminar, Variable Energy Cyclotron Centre, Kolkata, 28-29 October 2009.
4. Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.