Bileşen Tabanlı Yazılımlardaki Bağımlılıkların Azalması İçin Geliştirilen Kısıtlama Kontrolü Tasarım Kalıbı

Erdem Ergul[1], Ezgi Cankurtaran[2], Evren Çilden[3]

[1,2,3] Aselsan, REHİS Ankara, Türkiye
{eergul,ecankurtaran,ecilden}@aselsan.com.tr

Özet. Nesne yönelimli yazılımlarda nesneler arasındaki bağımlılıkların az ve yönetilebilir olması hedeflenmektedir. Bağımlılık faktörünün azalması, yazılıma; yeniden kullanılabilirlik (İng. Reusability), bakım yapılabilirlik (İng. Maintainability) ve anlaşılabilirlik (İng. Understandability) gibi yazılım özelliklileri kazandırarak yazılım kalitesini arttırmaktadır. Bu çalışmada, radar alanında (İng. Domain) geliştirilmiş bileşen tabanlı kullanıcı arayüzü yazılımında bileşenler arasındaki bağımlılıkları azaltmaya yönelik olarak geliştirilen Kısıtlama Kontrolü Tasarım Kalıbı anlatılmaktadır. Yazılımlarımızda bir bileşen tarafından üretilen kısıtlamanın başka bir bileşen tarafından kontrol edilip kullanılması ihtiyacı, bu bileşenler arasında bağımlılık kurma ihtiyacını ortaya çıkarmaktadır. Geliştirilen Kısıtlama Kontrolü Kütüphanesi ile uygulama içerisinde kısıtlama getiren durumların tanımlanması merkezi bir bileşen üzerinden açık olarak yapılmaktadır. Kısıtlama kontrol ihtiyacı duyan bileşenler, kısıtlamanın yapıldığı esas bileşene değil geliştirilen bu merkezi bileşene bağımlılık kuracağı için yazılımdaki toplam bağımlılık sayısı azalmıştır. Eklenen ve kontrol edilen yeni kısıtlamaların yönetiminin geliştirilen merkezi bileşenden yapılması hata riskini azaltmakta ve kodun anlaşılabilirliğini arttırmaktadır**.** Ayrıca, kütüphane çalışma zamanında, uygulama için mevcut bütün kısıtlamaların izlenmesine olanak vererek hata giderilmesine olanak sağlamaktadır. Geliştirilen tasarım kalıbı ile farklı alanlara özgü farklı yazılımlarda duyulan kısıtlama kontrolü ihtiyacı karşılanabilecektir. Kütüphane örnek bir yazılımda kullanılmış, giriş (İng. Fan-in) ve çıkış yelpazesi (Fan-out) metrikleri alınarak bileşenler arasındaki bağımlılıkların tasarım kalıbı uygulandıktan sonra azaldığı doğrulanmıştır.

Anahtar Kelimeler: Yazılım Tasarım Kalıbı, Kısıtlama Kontrolü Tasarım Kalıbı, Nesne Yönelimli Yazılım, Bileşen Tabanlı Yazılım, Bağlaşım Faktörü

Restriction Controller Design Pattern for Reducing Coupling of Component Based Software

Erdem Ergul[1], Ezgi Cankurtaran[2], Evren Çilden[3]

[1,2,3,4] Aselsan, REHİS Ankara, Türkiye
{eergul,ecankurtaran,ecilden}@aselsan.com.tr

In Object-oriented software, few and manageable dependencies between objects is aimed. A decrease in the dependency factor increases software quality by bringing in software attributes like reusability, maintainability and understandability. In this study, the Restriction Controller Design Pattern is explained, which is developed for reducing the dependencies between the components of a component-based user interface software that was developed for the radar domain. In our software, the need for a constraint generated by a component to be controlled and used by another component reveals a dependency between these components. With the development of the Restriction Controller Library, the cases that bring in restrictions in the application are declared explicitly in a central component. Since the components that need for controlling a restriction make a dependency to this central component instead of the specific component that puts the restriction, the total number of dependencies in the software has been reduced. The management of newly added and controlled restrictions are done in the developed central component, which reduces the risk of error and increases the understandability of code. Moreover, the library facilitates bug-fixing by enabling the run-time monitoring of all existing restrictions in the application. With the use of the developed design pattern, it will be possible to meet the need for controlling restrictions in various software for different domains. The library was used in a sample software, and the fan-in and fan-out metrics were taken to validate that the dependencies between the components had been reduced after the application of the design pattern.

Key Words: Software Design Patterns, Restriction Controller Design Pattern, Object Oriented Software, Component Based Software, Coupling

1 Giriş

Yazılım mühendisliği araştırma çalışmaları, yazılımın kalitesinin geliştirilmesine odaklanmaktadır. Yazılımı oluşturan bileşenler arasında kurulan bağımlılıklar, yazılımın tasarım kalitesini belirleyen faktörlerin başında gelmektedir [1, 2].

Bağımlılığı sayısallaştıran faktörlerden öne çıkanının bağlaşım faktörü olduğu literatürde görülmektedir. Bağlaşım, yazılımın bir modülünün değişiminin başka bir modülün değişimine etkisi olarak tanımlanmaktadır. Bu tanıma göre, yazılımda tekrarlayan kod parçalarının bulunduğu modüllerde ve birbirini kullanan modüllerde bağlaşım söz konusudur. Nesne yönelimli programlarda, yazılımda kod tekrarlarının bulunmasının aksine modüllerin birbirini kullanması, istenen ve beklenen bir durumdur. Yani, bağlaşım normal bir yazılım özelliği olup bu değerin kontrol altında tutulması beklenmektedir [3].

Bileşenlerin bazı işlevlerini yerine getirebilmesi için gerekli ön koşullara başka bileşenlerde üretilen bilgilerle karar verme ihtiyacı, radar kullanıcı arayüzü yazılım bileşenlerimiz arasındaki bağımlılığı arttıran bir etkendir. Yazılım senaryolarımızın işleyişi gereği, bileşenler başka bileşenlerin bazı işlevlerine kısıtlama getirebilmektedir. Örneğin, SST (Savunma Sistemi Teknolojisi) CİT (Cihaz İçi Test) sorgu mesajının başka bir dış arayüze gönderilmesi, SST sisteminin bağlı ve hazır olması ile ethernet gücünün açık olması kısıtlarına bağlı olarak gerçekleşmektedir.

Bu çalışmada, bileşenler arasında bağımlılığa neden olan bilgi akışının bileşenler arasında doğrudan değil, merkezi bir bileşen üzerinden sağlanması için Kısıtlama Kontrolü Kütüphanesi geliştirilerek bağımlılığın azaltılması hedeflenmiştir. Geliştirilen Kısıtlama Kontrolü Kütüphanesi ile uygulama içerisinde kısıtlama getiren durumların tanımlanması merkezi bir bileşen üzerinden açık olarak yapılmaktadır. Kısıtlama kontrol ihtiyacı duyan bileşenler, kısıtlamanın yapıldığı esas bileşene değil geliştirilen bu merkezi bileşene bağımlılık kuracağı için yazılımdaki toplam bağımlılık sayısı azalmıştır.

Çalışmanın 2. Bölümü'nde; Kısıtlama Kontrolü Tasarım Kalıbı aktarılmakta, 3. Bölümü'nde çalışmada uygulanan metot açıklanmakta ve Kısıtlama Kontrolü Tasarım Kalıbı'nı test etmek için geliştirilmiş örnek yazılım sunulmaktadır. 4. Bölüm'de geliştirilen tasarım kalıbının kullanıldığı ve kullanılmadığı Radar Kullanıcı Arayüzü Yazılımı'mızın giriş ve çıkış yelpazesi metrik ölçüm değerleri alınarak bağlaşım faktörleri karşılaştırılmakta, 5. Bölüm'de ise çalışmanın sonuçlarına yer verilmektedir.

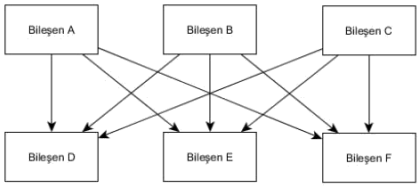
2 Kısıtlama Kontrolü Tasarım Kalıbı

Yazılımlarımızda bir işlevin ancak belirli koşullar sağlandığı zaman gerçekleşmesi sıklıkla karşılaşılan bir durumdur. Kendilerine kısıtlama getiren bu koşulların kontrol edilip sağlanması sonucunda bu işlevler yürütülmektedirler. Kısıt koyan ve bu kısıtları kontrol eden bileşenlerin farklı olması ise yazılımızdaki iç bağımlılığı arttırmaktadır. Örneğin radara ışıma başlat mesajının gönderilmesi işlevi, radarın gücünün açık olması ve antenin dönmesi kısıtlarına bağlıdır. Işıma başlat mesajını gönderen bileşen olan Işıma Bileşeni, Radar Güç Kontrol ve Anten bileşenlerinde tutulan durum bilgilerini kısıtlama kontrolü yaparken bu bileşenlere bağımlılık kurmaktadır.

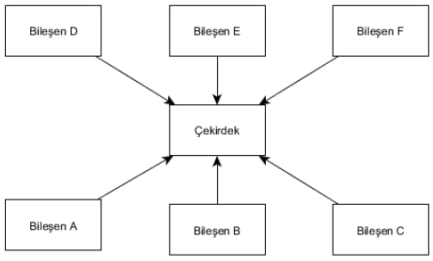
Kısıtlamalar bir merkezden kontrol edilmezse bileşenler arasında birçok bağımlılık kurulması gerekmektedir. Kurulan bağımlılıklar bakımı ve idamesi zor bileşenlerin oluşmasına neden olur. Literatürde yeniden kullanılabilir, bakımı kolay ve esnek yazılımların geliştirilmesinde hazır kalıplar olan tasarım desenlerine başvurulmaktadır. Çalışmada sunulan kısıtlama kontrolü tasarım kalıbı, tasarım desenlerinden aracı tasarım deseni (İng. Mediator) ile benzerlik göstermektedir. Her iki yaklaşımda da soyutlama ve merkezi bir bileşen üzerinden veri dağıtımının yapılması hedeflenir. Ancak kısıtlama kontrolü tasarım kalıbı daha çok kısıtlama kontrollerinin yapıldığı alana özelleşmiş bir yapıya sahiptir. Dâhili sınıf (İng. Inner Class) yapısı Annotation ile sağlanmıştır. Kütüphane olarak kullanılabilir olması esneklik kazandırmaktadır [7].

Senaryoya bağlı olarak "n" adet bileşenin koyduğu kısıtlamayı "m" adet bileşen kontrol ederek belirli bir işlevi yerine getirmektedir. En kötü durum senaryosunda (İng. Worst Case), Şekil 1'de sunulan "nxm" adet bağımlılık söz konusu olabilmektedir.

Kısıtlamaların tek bir merkez bileşenden kontrolü ise kısıt kontrolünü kolaylaştırdığı gibi bileşenler arasındaki bağımlılığı da azaltmaktadır. Merkezi kontrol bileşenin kullanıldığı tasarım kalıbında en kötü durum senaryosundaki bağımlılık sayısı Şekil 2'de sunulduğu gibi "n+m" adete düşmektedir.



Şekil 1. Kütüphaneden önce oluşan bağımlılık



Şekil 2. Kütüphaneden sonra oluşan bağımlılık

Şekil 1'de kötü durum senaryosuna göre bileşen A, B, C, D, E ve F arasında ayrı ayrı kurulan bağımlılıklar sunulmaktadır. A, B ve C bileşenleri belirli bir işlevi yerine getirmek için D, E ve F bileşenlerinin koyduğu kısıtları ayrı ayrı kontrol etmektedir. Bağımlılığın bu kadar çok olması, diğer bileşenlerin koyduğu kısıtların tek tek kontrol edilmesinde hata yapma riskini de ortaya çıkarmaktadır.

Şekil 2'de, D, E ve F bileşenlerinin ürettiği belirli bir kısıtlama bilgisine erişim için merkezi bir çekirdek bileşen üzerinden kısıtlamaların yönetimi sunulmaktadır. Kısıtlama kontrolü tasarım kalıbı merkezi bileşen üzerinden kullanıldığı için bağımlılık sayısı düşmektedir.

Tasarım kalıbının kullanıldığı yazılımda, kısıtlama koyan kaynak bileşenler ve bu kısıtlamayı kullanan bileşen birbirlerinin değişiminden bağımsız hale gelmektedir. Aynı kısıtlamayı farklı bir bileşen koyduğu zaman bu kısıtı kullanan bileşen bu değişiklikten etkilenmemektedir. Yeni bir bileşen aynı kısıtlama durumunu kontrol etmek istediğinde de bu bilgiyi hangi bileşenlerin ürettiğini bilmesine gerek kalmadan tek kaynak üzerinden gerekli bilgiyi çekebilmektedir.

Kısıtlama yöneticisi kullanıldığı durumda bir bileşenin farklı kısıtlama durumlarını kontrol etmek için yeni bağımlılıklar kurması gerekmemekte, kısıtlama yöneticisine kurduğu bağımlılık sayesinde uygulama genelindeki tüm kısıtlama durumlarını kontrol edebilmektedir.

Kısıtlama tanımlamalarının uygulamanın tasarım aşamasında yapılması gerektirmektedir. Kısıtlama tanımlayıcıları anlaşılır ifadeler içermeli ve her bileşenin erişimine açık olmalıdır. Çalışma zamanında bileşenler, kısıtlama koşulları sağlandığında bu kısıtlama tanımlayıcılarını kullanarak kısıtlama koyabilmekte ve bu koşullar ortadan kalktığında kendi koydukları kısıtlamayı kaldırmaktan sorumlu sayılmaktadır. Herhangi bir zamanda bir kısıtlamanın kullanıcısı aynı kısıtlama tanımlayıcılarını kullanarak basit bir sorgu ile belirli bir kısıtlama durumunun mevcut olup olmadığını kontrol edip buna göre işlevini gerçekleştirebilmektedir. Belirli bir kısıtlayıcı durumun oluşması için tek bir noktadan bu kısıtlamanın koyulmuş olması yeterlidir. Belirli bir kısıtlayıcı durumunun ortadan kalkması için ise aynı kısıtlama tanımlayıcısı ile kısıtlama koymuş olan her bileşenin kendi koyduğu kısıtlamayı kaldırmış olması gerekmektedir.

Kısıtlamayı koyan bileşen kısıtlamayı neden koyduğuna ilişkin bir ifade belirtmek zorundadır. Bu şekilde aynı kısıtlama farklı kaynaklar tarafından farklı gerekçelerle koyulabilmektedir. Kısıtlama yöneticisi yapılan kısıtlama sorgularında bulunan kısıtlamaların nedenlerini log arayüzünü gerçekleyen loglayıcı sınıfa iletebilmektedir. Böylece yapılan her sorgu sonucunda kısıtlama nedenleri otomatik olarak loglanabilmektedir. Ayrıca kısıtlama yöneticisi belirli bir anda uygulama genelinde var olan tüm kısıtlama durumlarını loglayabilmekte ve böylece hata ayıklamayı kolaylaştırmaktadır.

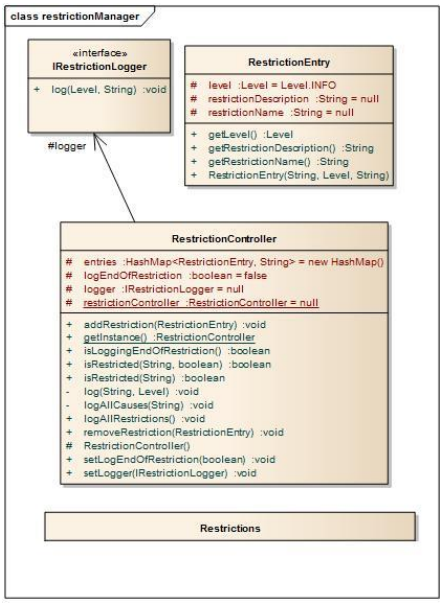
Kısıtlama yöneticisi kullanılmadığı durumda ve farklı kısıtlamaların kaynakları farklı olduğu takdirde ise bileşenlerin her yeni kısıtlama için yeni bağımlılıklar kurması gerekmektedir.

3 Uygulama

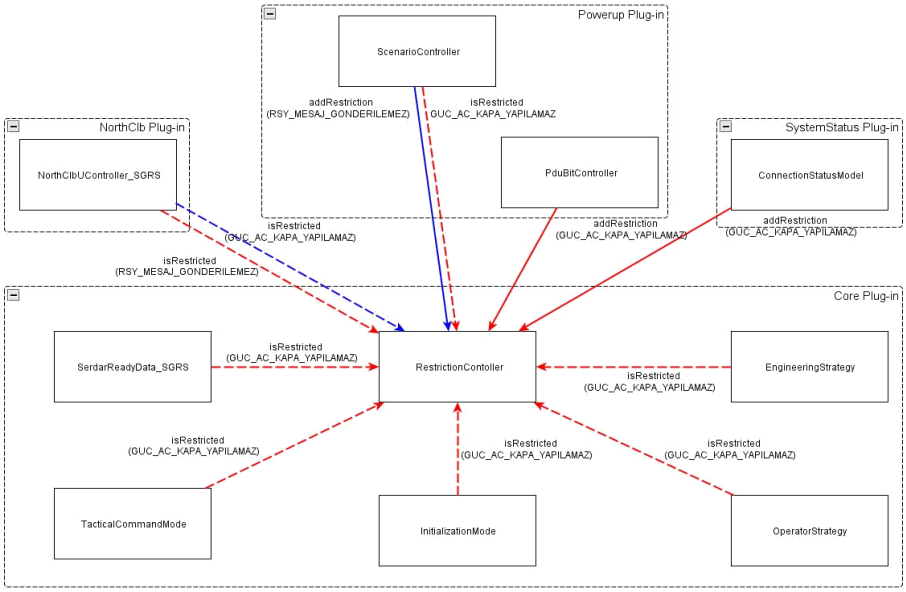
Yazılımda bileşenlerin bağımlılığının azaltılması için geliştirilen kütüphane, Radar kullanıcı arayüzü yazılımlarında kullanılmaya başlanmıştır. Kütüphanedeki sınıfların sahip oldukları metotlar ve sınıflar arasındaki ilişkiler Şekil 3'te sunulmaktadır:

RestrictionEntry; adı, açıklaması ve log seviyesi verilmiş bir kısıtlama nesnesini tanımlayan sınıftır. RestrictionController; kısıtlamanın eklenmesi ve kontrolü işlevlerini gerçekleştiren sınıftır. Ayrıca, loglama için yardımcı metotlar da içermektedir. Restrictions sınıfının içeriği, projeye özel ihtiyaçlar çerçevesinde doldurulacağından boş bırakılmıştır.

Şekil 4'te Radar Kullanıcı Arayüzü Yazılımı'mızda kullanılan Kısıtlama Kontrolü Tasarım Kalıbı'nın kısıt ekleme metodunu (addRestriction()) kullanan sınıflar sunulmaktadır. Kısıtın sorgulanmasına yönelik geliştirilen metodun çağrımı ise isRestricted() ile yapılmaktadır. isRestricted() metodu farklı parametreler alan adaş yordamlar olarak geliştirilmiştir. Adaş yordamların aldığı parametrelerden kısıt ismi (String restrictionName) ortak iken yordamlardan biri ek olarak loglama yapılıp yapılmayacağı parametresini (boolean doLogCauses) almaktadır.



Şekil 3. Kütüphanedeki sınıfların sahip oldukları metotlar ve sınıflar arasındaki ilişkiler



Şekil 4. Kısıt ekleme metodunun kullanımı

Kısıtlamalar, Restrictions sınıfı içerisinde özel bir Annotation (@Restriction) ile eklenen dâhili sınıflar (İng. inner class) olarak tanımlanmaktadır. Her kısıtlama türü için ismi kısıtlamayı tanımlayacak nitelikte olan ve herhangi bir değişken veya metodu olmayan bir JAVA sınıfı yani kısıtlama sınıfı tanımlanmaktadır. Kısıtlama sınıflarına ilişkin sınıf nesneleri yansıma (İng.Reflection) yöntemiyle elde edilerek kütüphaneye kısıtlama eklerken kullanılmaktadır. Bir kısıtlama sınıfı içerisinde başka bir kısıtlama sınıfı dâhili şeklinde tanımlandığında bu kısıtlamalar arasında mantıksal bir hiyerarşi kurulmaktadır. Şekil 5'te hiyerarşik yapıda tanımlanmış olan Restrictions sınıfından örnek bir kod parçası sunulmaktadır.

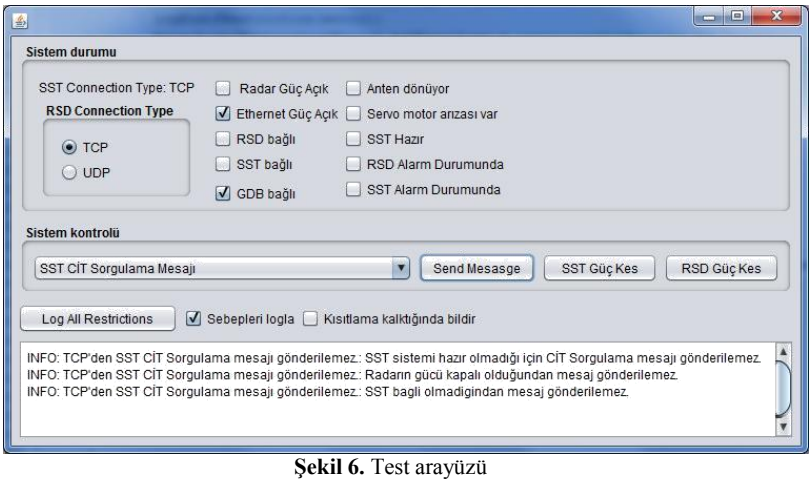
```
public class Restrictions {
   @Restriction("Mesaj gönderilemez.")
public class MESAJ_GONDERILEMEZ{
      @Restriction("UDP Mesajı gönderilemez.")
public class UDP{
         @Restriction("UDP'den RSD birimine mesaj gönderilemez.")
 public class RSD{
            @Restriction("UDP'den RSD Işıma Başlat mesajı gönderilemez.")
public class RSD_ISIMA_BASLAT_MESAJI{}
            @Restriction("UDP'den RSD Anten Döndür mesajı gönderilemez.")
 public class RSD_ANTEN_DONDUR_MESAJI{}
         }   }
      @Restriction("TCP mesajı gönderilemez.")
```

```
public class TCP{
        @Restriction("TCP'den SST birimine mesaj gönderilemez.")
public class SST{
            @Restriction("TCP'den SST CİT Sorgulama mesajı gönderilemez.")
public class SST_CIT_SORGULAMA_MESAJI{}
        }
        @Restriction("TCP'den RSD birimine mesaj gönderilemez.")
 public class RSD{
            @Restriction("TCP'den RSD Işıma Başlat mesajı gönderilemez.")
public class RSD_ISIMA_BASLAT_MESAJI{}
            @Restriction("TCP'den RSD Anten Döndür mesajı gönderilemez.")
public class RSD_ANTEN_DONDUR_MESAJI{}
        } } }
   @Restriction("Güç kesilemez")
 public class GUC_KESILEMEZ{
      @Restriction("SST biriminin gücü kesilemez")
 public class SST{}
      @Restriction("RSD biriminin gücü kesilemez")
 public class RSD{}
  }}
```

Şekil 5. Restrictions sınıfından örnek bir kod parçası

Yazılım geliştirmede kullanılan tümleşik geliştirme ortamı (İng. Integrated Development Environment) olarak kullanılan Eclipse, geliştiriciye kod tamamlama yetenekleri sunmaktadır. “Ctrl + Space” tuş kombinasyonları ile bir sınıfa ait dâhili sınıfların (İng. inner class) listesine kolaylıkla ulaşılabilmektedir. Bu sebeple bütün kısıtlama sınıflarının bir kapsayıcı sınıf içerisinde dâhili sınıflar halinde toplanması geliştiriciye büyük kolaylık sağlayacaktır. Üst seviyeden bir kısıtlama seçildikten sonra “Ctrl + Space” tuş kombinasyonları ile tümleşik geliştirme ortamı üst seviye kısıtın altındaki kısıtlama listesini de getirecektir. Bu şekilde ilerleyerek kısıtlama hiyerarşisinde gezinip istenilen alt seviye kısıtlama sınıfına kolaylıkla erişilebilmektedir.

Kütüphaneyi test etmek için test yazılımı geliştirilmiştir. Geliştirilen test yazılımının kullanıcı arayüzleri Şekil 6’da sunulmaktadır. Şekil 6’da sunulan senaryoda, SST CİT Sorgulama mesajı gönderilmesi için SST Hazır ve SST bağlı kısıtları isResricted() metodu ile sorgulanmıştır. SST Hazır ve SST Bağlı kısıtları söz konusu olduğu için mesaj gönderimi engellenmiş ve söz konusu durum log penceresinde operatöre sunulmuştur.

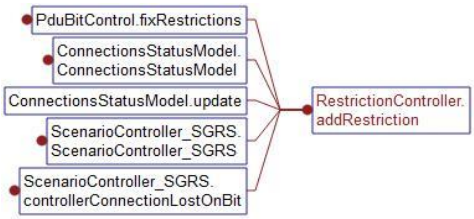


Şekil 6. Test arayüzü

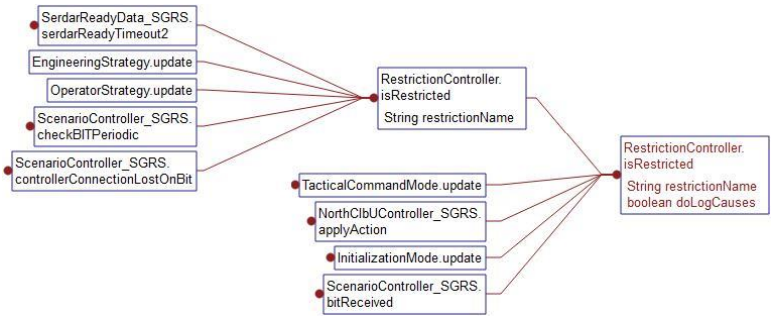
4 Ölçüm

Kısıtlama Yöneticisinin bileşenler arası bağımlılıklara etkisinin incelenebilmesi için kütüphanenin kullanılmış olduğu bir proje üzerinde bağımlılık analizi yapılmıştır. Bağımlılık analizi için bağlaşım faktörünü sayısallaştıran giriş ve çıkış yelpaze metrikleri kullanılmıştır [1, 4-6]. Örnek projede iki farklı kısıtlama tanımı yapılmıştır:

- Güç açma / kapama işlemi yapılamaz
- RSY birimine mesaj gönderilemez



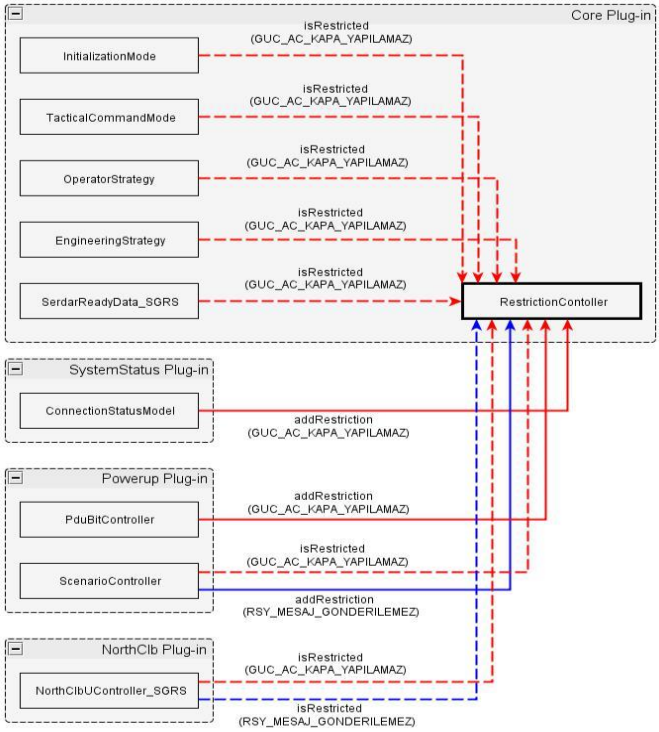
Şekil 7. addRestriction() metodunu kullanan sınıflar



Şekil 8. isRestricted() metodunu kullanan sınıflar

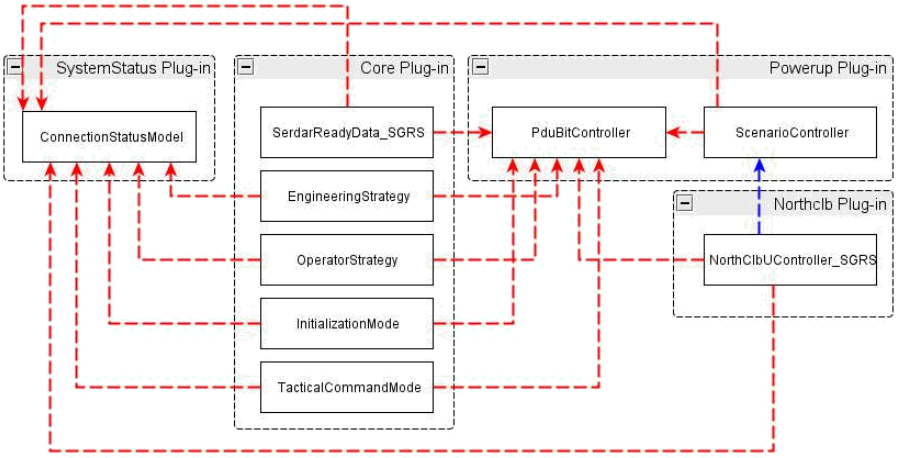
Şekil 7 ve 8 incelendiğinde, bir sınıfın RSY birimine mesaj gönderilemez kısıtını eklerken iki sınıfın Güç açma / kapama işlemi yapılamaz kısıtını eklediği görülmektedir. RSY birimine mesaj gönderilemez kısıtını bir sınıf kontrol ederken Güç açma / kapama işlemi yapılamaz kısıtını yedi sınıf kontrol etmektedir. İki farklı kısıtlama durumunu birden kontrol eden, iki sınıf olduğu için kütüphanenin kullanımı sonrasında toplam bağımlılık sayısı azalmaktadır. Kısıtlama yöneticisi kullanıldığı durumda bir kısıt için bağımlılık sayısı m + n olduğu için toplam bağımlılık sayısı (1+1)+(2+7) - 2 = 9 olarak hesaplanabilir.

Şekil 9'da bileşenlerin eklediği ve kontrol ettiği bağımlılıklar detaylı sunulmaktadır:



Şekil 9. Bileşenlerin eklediği ve kontrol ettiği bağımlılıklar

Kısıtlama yöneticisi kullanılmasaydı en kötü durumda tüm bileşenler ihtiyaç duydukları kısıtlama bilgisine kaynak bileşene doğrudan bağımlılık kurarak erişecekleri için Şekil 10'da sunulan yapıda bağımlılıkların kurulması gerekecekti:



Şekil 10. Kütüphanenin kullanılmadığı yapıda bağımlılıklar

5 Sonuçlar

Bu çalışmada, bileşenler arasında bağımlılığa neden olan bilgi akışının bileşenler arasında doğrudan değil, merkezi bir bileşen üzerinden sağlanması için geliştirilmiş olan Kısıtlama Kontrolü kütüphanesi sunulmuştur. Bu kütüphane ile kısıtlama getiren durumların tanımlanması merkezi bir bileşen üzerinden açık olarak yapılmıştır. Kısıtlama kontrol ihtiyacı duyan bileşenler, kısıtlamanın yapıldığı esas bileşene değil geliştirilen merkezi bileşene bağımlılık kurarak yazılımdaki toplam bağımlılık sayısını azaltmıştır. Bu kullanım, kaynak kodun anlaşılabilirliğini ve yeniden kullanılabilirliğini önemli ölçüde kolaylaştırmaktadır [1]. Kısıtlama getiren durumların kolaylıkla loglanabilmesi, bir işleve ait ön koşulun hangi kısıtlama nedeniyle sağlanamadığının kayıtlardan kolaylıkla öğrenilebilmesi, yazılımımızın daha kolay bakım yapılabilirliğini sağlamıştır. Çalışma, örnek bir projede kullanıma alınmıştır. İlerleyen dönemlerde kütüphanenin kullanımının diğer radar kullanıcı arayüzü yazılımlarında da yaygınlaştırılması hedeflenmektedir. Kütüphane, sadece radar kullanıcı arayüzü alanına özel olmayıp, farklı alanlarda benzer ihtiyaçlara sahip yazılımlarda da kullanılabilir niteliktedir.

Referanslar

1. Cankurtaran E., Cilden E., and Tarhan A., "Bileşen Tabanlı ve Ürün Hattı Yazılım Geliştirme Yaklaşımlarında Yeniden Kullanılabilirlik Metrikler," in *Turkish National Software Engineering Symposium (In Turkish: Ulusal Yazılım Mühendisliği Sempozyumu (UYMS))*, Çanakkale, 2016.
2. Akdur D. and Özdemir Ç., "The Impacts of Reusable and Configurable Modules on Software Quality in Real-Time Embedded Systems: Radar Utility Libraries

(In Turkish: Gerçek Zamanlı Gömülü Sistemlerde Yeniden Kullanılabilir ve Yapılandırılabilir Yazılımların Kaliteye Etkisi: Radar Projeleri Destek Kütüphaneleri)," in *8th Turkish National Software Engineering Symposium (In Turkish: Ulusal Yazılım Mühendisliği Sempozyumu (UYMS))*, Cyprus, 2014, pp. 177-186.

3. Cataldo M., Mockusi A., Roberts J. A., and Herbsleb J. D., "Software Dependencies, Work Dependencies, and Their Impact on Failure," *IEEE Transactions on Software Engineering,* vol. 35, 2009.
4. Henry S. and Kafura D., "Software Structure Metrics Based on Information Flow," *IEEE Transactions on Software Engineering,* vol. 7, 1981.
5. Yu-ying W., Qing-shan L., Ping C., and Chun-de R., "Dynamic fan-in and fan-out metrics for program comprehension," *Journal of Shanghai University,* vol. 11, 2007.
6. Hamza S., Sadou S., and Fleurquin R., "Measuring Qualities for OSGi Component Based Applications," presented at the 13th International Conference on Quality Software, 2013.

7. Almarza F.A., Ponce H.R., Lopez J., “Software Component Development based on Mediator Pattern Design: the Interactive Graphic Organizer Case”, IEEE Latin America Transaction, 2011.