Görev Kritik ve Gömülü Sistemler için T[5]D Uyumlu Bir Hata Yönetimi Altyapısı Tasarımı ve Gerçeklemesi

Metin Tekkalmaz[1] Eda Gürler[2] Mustafa Dursun[3]

[1,2,3] Radar Elektronik Harp ve İstihbarat Sistemleri (REHİS) Grubu, ASELSAN A.Ş., Ankara

[1] e-posta: tkalmaz@aselsan.com.tr [2] e-posta: tverdi@aselsan.com.tr [3] e-posta: mdursun@aselsan.com.tr

Özetçe

Bu bildiride, görev kritik ve gömülü yazılımlarda kullanılmak üzere gerçeklenmesi büyük ölçüde tamamlanmış bir hata yönetimi altyapısı tasarımı sunulmaktadır. Sunulan hata yönetimi altyapısı, kullanıldığı sistemlerde yazılım ve donanım kaynaklı arızaları mümkün olan en erken safhada tespit ederek arızanın etkilerini en aza indirgemeyi hedeflemektedir. Hata yönetimi altyapısı kural ve yapılandırma dosyaları ile ihtiyaçlara göre uyumlandırılabilir özelliğe sahiptir. Ayrıca tasarlanan arakatman sayesinde ana işlevleri gerçekleyen bileşenler, birbirlerinin yerlerinden, sayılarından ve aralarındaki haberleşme yöntemlerinden bağımsız bir şekilde çalışabilmektedirler. Hata yönetimi altyapısı, bu özelliği sayesinde dağıtık konuşlandırmaya da olanak sağlamaktadır. Bildiride anlatılan tasarım daha önce sunulan bir hata yönetimi kılavuz mimarisi olan T[5]D'yi referans almaktadır.

1. Giriş

Gereklerin eksik tanımlanması, tasarımın ya da gerçeklemenin kusurlu olması gibi nedenler dolayısı ile yazılımlar hata içerebilirler. Her ne kadar yazılım geliştirme faaliyetinin farklı aşamalarındaki test ve doğrulama faaliyetleri bu hataları ortaya çıkarmayı amaçlasa da yazılımların günümüzde ulaştıkları büyüklük düşünüldüğünde hata içermeyen bir yazılım ürünü elde etmek pratikte olanaksızdır. Yazılım içerisinde yer alan bu hatalar çalışma zamanında çeşitli arızalara yol açabilir. Bu durum, arızanın tipine ya da yazılımın türüne göre önemsiz olabileceği gibi yazılımın işlevini yerine getirememesi, maddi zarar, hatta can kaybı sonuçlarını da doğurabilir. Yazılımlar için hata yönetimi ihtiyacı bu noktada ortaya çıkmaktadır. Hata yönetimi, arızaların olumsuz etkilerini ortadan kaldırmak ya da bu mümkün olmuyorsa en aza indirmek amacı ile sistemden toplanan bilgiyi işleyerek oluşması beklenen ya da oluşan arızaları tahmin ya da tespit eder, kaynağını bulur ve nihayetinde önleyici ya da düzeltici faaliyetlerde bulunur.

ASELSAN Görev Yazılımları Müdürlüğü bünyesinde geliştirilen radar ve elektronik harp yazılımları için hata yönetimi ihtiyacını karşılamak amacı ile Radar Elektronik Harp Referans Modeli'nde (REFoRM) [1] hata yönetimi işi bir servis olarak yer almıştır. Bir sonraki adım olarak düşünülebilecek ve ODTÜ Bilgisayar Mühendisliği Bölümü ile birlikte yürütülen hata yönetimi kılavuz mimarisi çalışmasında ise farklı projelerde yeniden kullanılabilmek üzere tasarlanan bileşenlerden oluşan T[5]D mimarisi ortaya konulmuştur [2]. T[5]D, farklı gömülü yazılım projelerinde kullanılabilmek üzere, hız, haberleşme, dağıtıklık ve özelleşmiş donanımlar ile birlikte çalışma gibi konulardan kaynaklanan ek zorluk ve karmaşıklığı dikkate alarak bir ürün hattı yaklaşımına [3] uygun olarak geliştirmiştir.

Bu makalede görev kritik ve gömülü sistemler için geliştirilmiş ve bahsi geçen T[5]D kılavuz mimarisini temel alan bir hata yönetimi altyapısı tasarımı ve gerçeklemesi anlatılmaktadır. T[5]D'nin uygulama alanından kaynaklanan gerek ve kısıtları açıklanan tasarımda da göz önünde bulundurulmuş, aynı odak noktalarının alt seviyelerde de adreslenmesine dikkat edilmiştir. Bu kapsamda farklı projeler dâhilinde yeniden kullanımın mümkün olduğu, modüler ve uyumlandırılabilir bileşenler tasarlanmış ve gerçeklenmiştir. Hata yönetimi temel işlevlerini yerine getirmek üzere belirtilerin izlenmesi, arızanın varlığının algılanması (tespit), arızanın asıl kaynağının bulunması (teşhis), arızanın düzeltilmesi ve gerektiğinde tüm bu faaliyetlerin raporlanması işleri kurallar aracılığı ile yapılandırılabilir bileşenler halinde ortaya konulmuştur. Bu temel bileşenlerin farklı sistem mimarileri ve haberleşme kısıtlarına göre gerektiğinde dağıtık olarak konuşlandırılabilmelerine olanak sağlamak amacıyla da arakatmana ihtiyaç duyulmuştur. Hazır arakatman ürünlerinin, geliştirilen gömülü sistemlerdeki heterojen haberleşme altyapılarının tamamını desteklememeleri, ayrıca genel yazılımlar olmaları sebebi ile yazılım büyüklüğüne ve çalışma zamanı performansına olumsuz etkileri dolayısı ile hazır arakatman ürünleri kullanmak yerine ihtiyaca uygun arakatman bileşenleri geliştirilmiştir. Diğer taraftan, ortaya konulan arakatman bileşen tasarımı, ihtiyaç durumunda, hazır arakatman ürünlerinin kolay bir şekilde kullanıma alınmasına imkan sağlamaktadır.

Makalenin geri kalanında sunum şu şekildedir: 2. bölüm hata yönetimi işinin temel işlevlerini yerine getiren bileşenleri, 3. bölüm ise temel bileşenlerin dağıtık çalışmasına olanak sağlayan arakatman bileşenlerini anlatmaktadır. 4. bölümde bir dağıtık konuşlandırma örneği üzerinden temel bileşenlerin ve arakatman bileşenlerinin birlikte kullanımı açıklanmaktadır. Son bölümde ise ortaya konulan sistem genel olarak değerlendirilmekte ve hedeflenen sonraki çalışmalar verilmektedir.

2. Temel Bileşenler

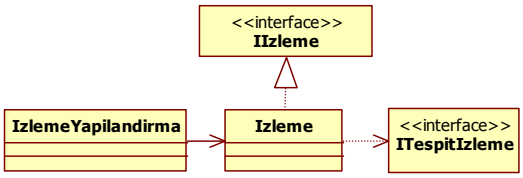
Sunulan hata yönetimi altyapısı tasarımına göre, bilginin toplanması, işlenmesi, filtrelenmesi, gerektiğinde biriktirilmesi, saklanması, ihtiyaç varsa raporlanması gerekmektedir. Bu görevler, alt başlıklarda açıklanan, geliştirilmesi tamamlanmış olan bileşenler tarafından gerçekleştirilmektedir.

2.1. Belirti İzleme

Belirtiler hata yönetimi işinin temel girdileridir ve koşan koda ait işlemlerin durumları, işlemci ve bellek kullanım miktarları, sabit disk doluluğu gibi bilgilerdir. Çalışması sistem için kritik

olan birimlerden gelen "hayattayım" mesajları da belirti olarak kabul edilir. Belirtilerin, tiplerine göre, belli sınırlar dışına çıkmaları, sıklıklarının değişmesi, kesilmeleri gibi durumlar oluşmuş veya ileride oluşması olası arızalara işaret ediyor olabilir.

Sunulan tasarımda belirtiler Belirti İzleme bileşenleri tarafından takip edilir ve izleme sonucunda üretilen belirtiler, belirti kaynağı, belirti tipi ve belirti içeriği bilgilerini barındırır. Belirti İzleme bileşenleri görevlerini yerine getirebilmeleri için izlemenin yapılacağı noktalarda konuşlandırılırlar. Her bir bileşen kendi yapılandırma dosyasına sahiptir ve bu dosyadan okunan eşik, izleme periyodu gibi bilgileri kullanarak topladığı belirtileri bağlı olduğu Tespit bileşenine aktarır.



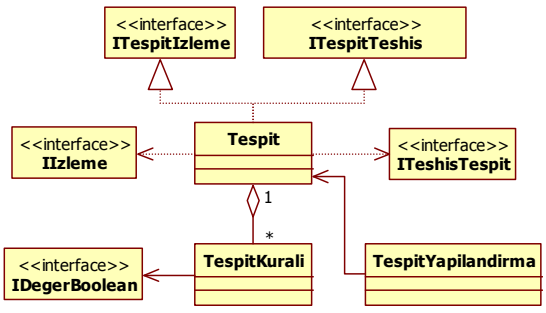
Şekil 1: İzleme Bileşeni Tasarımı

Şekil 1'de bir İzleme sınıfının sahip olduğu arayüzler gösterilmektedir. Izleme sınıfı yapılandırma dosyasını kullanan IzlemeYapilandirma sınıfı tarafından yapılandırılmaktadır. IIzleme arayüzü, tespit tarafından yapılacak sorgular için gerçeklenmektedir. Tespit, izlemeden, belirti veya hangi belirtilerin toplandığı bilgisini sorgulayabilir. Toplanan belirtiler, ITespitIzleme arayüzünden tespite gönderilir.

2.2. Tespit

Belirti İzleme bileşenleri tarafından aktarılan belirtiler, Tespit bileşenlerinde, tespit kurallarına göre değerlendirilir. Bu değerlendirme sonucunda, arızanın oluştuğu veya ortadan kalktığı kararı verilir. Tespitin girdileri belirti, çıktıları arıza biçemindedir. Tasarımda arıza, arıza tipini ve belirtide olduğu gibi hangi kaynağa ait olduğu bilgisini içerir.

Bir kaynakta çıkabilecek arızalar ve bunlara işaret edecek belirtiler, tespit kurallarının oluşturulabilmesi için önceden biliniyor olmalıdır.



Şekil 2: Tespit Bileşeni Tasarımı

Şekil 2'de bir Tespit sınıfının sahip olduğu arayüzler verilmektedir. Ayrıca, bu sınıfın kural ve yapılandırma dosyalarıyla arayüzünü sağlayan TespitKurali ve TespitYapilandirma sınıflarıyla olan ilişkileri gösterilmektedir.

Tespit bileşeninin, hem kendisine belirti gönderen İzleme bileşeniyle, hem de kendisinden tespit sonucu bekleyen Teşhis bileşeniyle ilişkisi vardır.

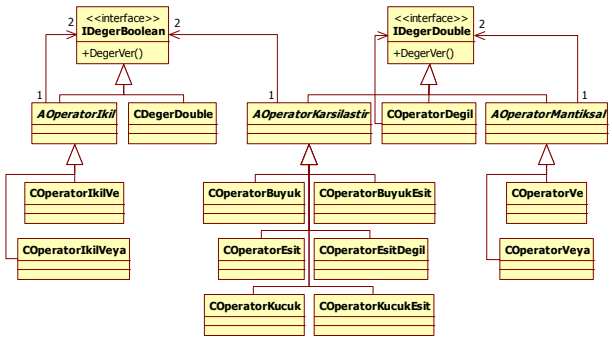
Tespit, kendisine belirti gönderecek olan İzleme bileşeni için ITespitIzleme arayüzünü, kendisinden tespit sonucu sorgulayacak olan Teşhis bileşeni için de ITespitTeshis arayüzünü gerçeklemektedir.

Tespit, ihtiyaç duyduğu zaman kendisine bağlı olan İzleme bileşenlerinden belirti sorgulayabilir ve bu sorgusunu IIzleme arayüzü üzerinden gerçekleştirir. Tespit, belirti ve tespit kurallarını kullanarak elde ettiği tespit sonuçlarını, ITeshisTespit arayüzü üzerinden, bağlı olduğu Teşhis bileşenine aktarır.

Sınıf seviyesinde tasarımı verilen Tespit bileşeni pIzlemeTespit portu üzerinden IIzleme arayüzüne gereksinim duymakta, ITespitIzleme arayüzünü sağlamaktır, pTespitTeshis portu üzerinden ise ITeshisTespit arayüzüne gereksinim duymakta, ITespitTeshis arayüzünü sağlamaktadır. Diğer temel bileşenlerde de gerçeklenen ve ihtiyaç duyulan arayüzler, benzer şekilde port kullanımıyla uygulamaya geçirilmiştir.

Tespit kuralları, tasarımda belirtilen gramer tanımına uygun olarak yazılan sebep-sonuç ifadeleridir. Kurallara ait gramer aşağıda tanımlanmıştır. Tanımlama Extended Backus-Naur Form [4] biçeminin bir türevi ile yapılmıştır.

tespit_ifade → *tespit_koşul* ":" *eylem*
eylem → *arıza*
arıza → "arıza(" *kaynak* "," *tamsayı* ")"
kaynak → "kaynak(" *tamsayı* ("," *tamsayı*)* ")"
tespit_koşul → (("(" *tespit_koşul* ")") | *işlem*) (*mantık_işleci tespit_koşul*)*
işlem → *işlenen koşul_işleci işlenen*
işlenen → (("(" *işlenen* ")") | *hexsayı* | *reelsayı* | *belirti*) (*ikil_işleci işlenen*)*
belirti → "belirti(" *kaynak* "," *tamsayı* ")"
mantık_işleci → "&&" | "||"
koşul_işleci → "<" | ">" | "<=" | ">=" | "!=" | "=="
ikil_işleci → "&" | "|"



Şekil 3: Kurallar için Sınıf Çizgesi

Bu gramere uygun yazılan kurallar bir LL ayrıştırıcı [5] ile ayrıştırıldıktan sonra Şekil 3'te gösterilen sınıf çizgesinde verilen sınıflara ait nesnelerin oluşturduğu ve ilişkiler göz önüne alındığında ağaç biçimli bir yapıda saklanır. Ağacın kök nesnesi Şekil 2'de gösterildiği üzere IDegerBoolean

arayüzünü gerçeklemiş bir sınıfa ait nesnedir. Kural dosyası, yazılım çalışmaya başladığı zaman bir kez okunur ve ayrıştırılır, kuralın çalıştırılması bahsi geçen yapı üzerinden gerçekleştirilir.

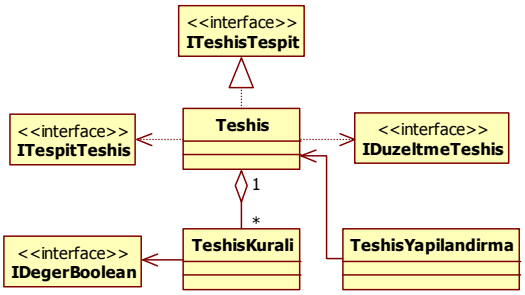
Bu gramer kullanılarak yazılabilecek olan tespit kuralları aşağıda bir örnek ile açıklanmaktadır:

```
(belirti(kaynak(1), 4) & 0x03 != 0x00) &&
(belirti(kaynak(1,3), 6) > 60) :
arıza(kaynak(1), 2)
```

Bu tespit kuralına göre, kaynak(1)'den gelen 4 numaralı belirtinin en az önemli iki bitinin 0'dan farklı ve kaynak(1,3)'ten gelen 6 numaralı belirtinin değerinin 60'dan büyük olması, kaynak(1)'de 2 numaralı arıza oluştuğu anlamına gelmektedir.

2.3. Teşhis

Tespit bileşenleri tarafından aktarılan tespit sonuçları, Teşhis bileşeninde, teşhis kurallarına göre değerlendirilir. Teşhisin hem girdileri hem de çıktıları arıza biçemindedir. Ancak girdiler arızanın varlığını, yani tespit sonuçlarını ifade ederken çıktılar arızanın asıl kaynağını, yani teşhis sonuçlarını ifade eder. Teşhis sonuçlarının üretilmesi için genel olarak birden fazla tespit sonucu bir arada değerlendirilir.



Şekil 4: Teşhis Bileşeni Tasarımı

Şekil 4'te bir Teşhis sınıfının sahip olduğu arayüzler verilmektedir. Ayrıca, bu sınıfın kural ve yapılandırma dosyalarıyla arayüzünü sağlayan TeshisKurali ve TeshisYapilandirma sınıflarıyla olan ilişkileri gösterilmektedir.

Teşhis, kendisine tespit sonucu gönderecek olan Tespit bileşeni için ITeshisTespit arayüzünü gerçeklemektedir. Düzeltme bileşeninin Teşhis'ten teşhis sonucu sorgulaması gibi bir ihtiyaç olmadığı için bu amaca yönelik bir arayüz eklenmemiştir.

Teşhis, ihtiyaç duyduğu zaman kendisine bağlı olan Tespit bileşenlerinden tespit sonucu sorgulayabilmektedir. Teşhis, bu sorgusunu ITespitTeshis arayüzü üzerinden gerçekleştirir. Teşhis, kendisine gönderilen tespit sonuçlarını ve teşhis kurallarını kullanarak elde ettiği teşhis sonuçlarını, IDuzeltmeTeshis arayüzü üzerinden, bağlı olduğu Düzeltme bileşenine aktarır.

Teşhis kurallarının yazılabilmesi için Tespit gramerine aşağıdaki ekleme ve değişiklikler yapılmıştır:

teşhis_ifade → *teşhis_koşul* ":" *eylem*
koşul_temel → "("*teşhis_koşul* ")" | *arıza* | *koşul_değil*
teşhis_koşul → *koşul_temel* (*mantık_işleci*)* *teşhis_koşul*
koşul_değil → "!"*teşhis_koşul*

Aşağıda, bir teşhis kuralı örneği verilmiştir:

```
arıza(kaynak(0,1), 1) || arıza(kaynak(0,2), 2)
: arıza(kaynak(0), 3)
```

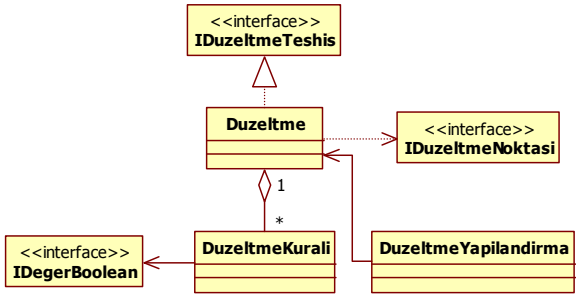
Bu teşhis kuralına göre, kaynak(0,1)'de 1 numaralı veya kaynak(0,2)'de 2 numaralı arızalardan herhangi birinin bulunması, arızanın temel sebebinin kaynak(0)'daki 3 numaralı arıza olduğunu göstermektedir.

2.4. Düzeltme

Teşhis bileşenleri tarafından aktarılan teşhis sonuçları, Düzeltme bileşeninde, düzeltme kurallarına göre değerlendirilir. Düzeltme çıktıları, yürütülmesi gereken düzeltme faaliyetlerini ifade eder. Düzeltme faaliyetleri, sistemde dağıtık olarak konuşlanmış Düzeltme Noktalarında gerçekleştirilir.

Şekil 5'te Düzeltme sınıfının sahip olduğu arayüzler verilmektedir. Ayrıca, bu sınıfın kural ve yapılandırma dosyalarıyla arayüzünü sağlayan DuzeltmeKurali ve DuzeltmeYapilandirma sınıflarıyla olan ilişkileri gösterilmektedir.

Düzeltme, kendisine teşhis sonucu gönderecek olan Teşhis bileşeni için IDuzeltmeTeshis arayüzünü gerçeklemektedir. Ayrıca Düzeltme, verdiği düzeltme kararlarını uygulatmak amacıyla IDüzeltmeNoktasi arayüzünü kullanır.



Şekil 5: Düzeltme Tasarımı

Düzeltme kurallarının yazılabilmesi için Teşhis gramerine aşağıdaki ekleme ve değişiklikler yapılmıştır:

düzeltme_ifade → *teşhis_koşul* ":" *düzeltme_eylem*
düzeltme_eylem → *düzeltme*
düzeltme → "düzeltme(" *kaynak* "," *tamsayı* ")"

Aşağıda bir düzeltme kuralı örneği verilmiştir:

```
arıza(kaynak(0, 9), 0) && !arıza(kaynak(0, 4),
1) : düzeltme(kaynak(0), 7)
```

Bu düzeltme kuralına göre, kaynak(0,9)'da 0 numaralı arızanın bulunması ve kaynak(0,4)'te 1 numaralı arızanın bulunmaması durumunda, kaynak(0) üzerinde 7 numaralı düzeltme faaliyeti uygulanmalıdır.

2.5. Raporlama

Tespit ve teşhis sonuçları istenirse kullanıcıya raporlanabilir. Bu yetenek için Raporlama bileşeni tasarlanmıştır. Raporlama, tespit ve teşhis sonuçlarını, zaman bilgisiyle birlikte, değişiklik oldukça veya istek üzerine kullanıcıya bildirir.

Raporlama sınıfı, Tespit ve Teşhis bileşenlerinin ürettiği sonuçları alabilmek amacıyla hem ITeshisTespit hem de IDuzeltmeTeshis arayüzlerini gerçekler.

Yapılandırma dosyasında, sistemde çıkması olası arızaların bir altkümesi yapılandırma dosyaları aracılığı ile Sistem İçi Test kapsamına alınabilir. Bu sayede hata yönetimi altyapısı, Sistem İçi Test sonuçlarının elde edilmesi ve kullanıcıya sunulması amacıyla da kullanılabilir.

3. Arakatman Bileşenleri

Önceki bölümde ayrıntıları verilen hata yönetimi altyapısı temel bileşenleri aynı arayüzü gerçekleyen en fazla bir komşu bileşenle (1'e 1 ilişki), aynı bellek adres alanı içerisinde haberleşecek şekilde (doğrudan metot çağrımı) tasarlanmışlardır. Ancak, mevcut tasarımlarının aksine bu bileşenlerin hem aynı arayüzü gerçekleyen birden fazla bileşenle birlikte çalışabilmeleri, hem de farklı adres alanlarında dağıtık olarak kullanılabilmeleri gerekmektedir. Bu gereklerin temel bileşenler kapsamında ele alınması yerine arakatman bileşenlerinin kullanımı öngörülmüştür.

Arakatman yaklaşımının beraberinde getirdiği modülerlik sayesinde projelere özel ürün sağlamak daha kolay hale gelmiştir. Bu yaklaşımın açık/kapalı ilkesine (Open/Closed Principle [6]) uygunluğu sayesinde temel bileşenler ve mevcut arakatman bileşenleri üzerinde bir değişikliğe ihtiyaç duyulmaksızın temel bileşenlerin etkileşimleri açısından farklı yöntemlerin altyapıya eklenmesi mümkün hale gelmiştir.

Arakatman bileşenleri temel işlevlerin aşağıdaki 3 açıdan soyutlanmalarını sağlamaktadır:

1. Bir temel bileşen, haberleşeceği temel bileşenin yalnızca arayüzünden haberdardır ve bu temel bileşenin *nerede konuşlandığını* bilmemektedir.
2. Bir temel bileşen, haberleşeceği temel bileşenin *kaç örneğine* mesaj gönderdiğini bilmemektedir.
3. Bir temel bileşen, haberleşeceği temel bileşenin *hangi örneğine* mesaj gönderdiğini bilmemektedir.

Belirtilen soyutlama gereklerini sağlayabilmek için arakatman bileşenleri 3 grup bileşen türü altında tasarlanmıştır. Bunlar haberleşme, çoklayıcı ve anahtarlama bileşen türleridir. Her bir bileşen türü Vekil (Proxy) tasarım kalıbını [7] temel alacak şekilde tasarlanmıştır.

3.1. Haberleşme Bileşenleri

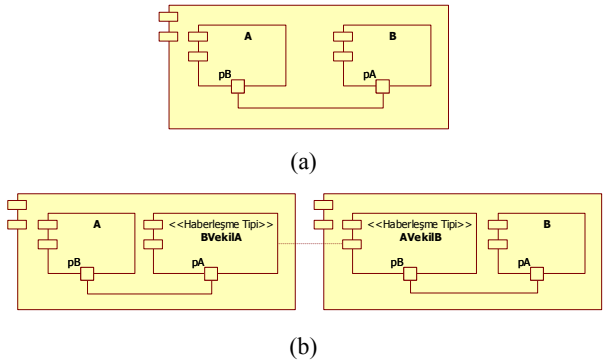
Haberleşme bileşenleri temel bileşenlerin dağıtık ortamlarda haberleşme ihtiyacının karşılanmasını sağlamaktadır. Haberleşme bileşenleri, haberleşme ve vekil kısmından oluşur. Haberleşme kısmı haberleşme yöntemi (TCP, UDP, seri kanal, paylaşımlı bellek v.b.) üzerinden haberleşmeyi sağlamaktadır. Vekil kısmı ise temel bileşenleri konuşlandırma bilgisi açısından dağıtıklıktan soyutlamaktadır.

Uygulamada haberleşme bileşeni her temel bileşen için ve haberleşme yöntemi için çeşitlendirilmiştir. Bu kapsamda haberleşme bileşenlerinin adlandırılması şu şekilde yapılmaktadır:

(Vekili olduğu bileşen adı) + "Vekil" + (Kullanan Bileşen Adı)

Bir haberleşme bileşeni, vekili olduğu temel bileşenin sunulan ve gereken arayüzlerini aynı şekilde uygular. Örneğin A temel bileşeni ve B temel bileşeninin haberleşebilmesi için BVekilA haberleşme bileşeninin ve karşılığı olan AVekilB haberleşme bileşeninin kullanılması gerekir. BVekilA, pB portu üzerinden A temel bileşenin B temel bileşeninden beklediği arayüzü sunarken, B temel bileşeninin A temel bileşeninden beklediği arayüzü kullanır. AVekilB'ye ait pA portu için bunun tersi geçerlidir.

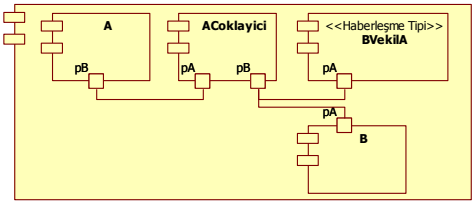
Aynı bellek adres alanı içindeki temel bileşenlerin haberleşme yapısı Şekil 6 (a)'da gösterilmiştir. Farklı bellek alanlarındaki temel bileşenlerin BVekilA ve AVekilB haberleşme bileşenleri kullanımı ile haberleşmesi ise Şekil 6 (b)'de verilmiştir.



Şekil 6: (a) Temel Bileşenlerin İlişkisi, (b) Haberleşme Bileşeni Kullanımı

3.2. Çoklayıcı Bileşenler

Çoklayıcı bileşenler bir temel bileşenin haberleşeceği temel bileşenin birden fazla örneğine mesaj göndermesini sağlayan arakatman bileşenleridir. Bu amaçla mesaj gönderecek temel bileşenin ilgili bağlantı sayısını çoklar.

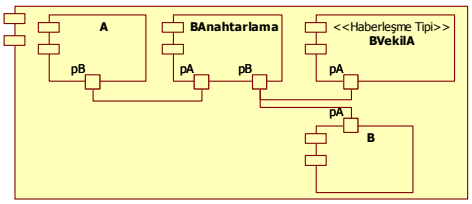


Şekil 7: Çoklayıcı Bileşeni Kullanımı

Bu arakatman bileşeninin kullanımı Şekil 7'de gösterilmiştir. ACoklayici pA portu üzerinden A temel bileşeninin B temel bileşeninden beklediği arayüzü sunarken, B temel bileşeninin A temel bileşeninden beklediği arayüzü kullanır. Bu port tek bir bağlantıya izin verir. Çoklu bağlantıya izin veren pB portu üzerinden ise B temel bileşeninin A temel bileşeninden beklediği arayüzü sunarken, A temel bileşeninin B temel bileşeninden beklediği arayüzü kullanır.

3.3. Anahtarlama Bileşenleri

Bir temel bileşenin, haberleşeceği temel bileşenin birden fazla örneği ile etkileşim içerisinde olmasına rağmen, yalnızca o an için uygun bir tanesine mesaj göndermesi ihtiyacı bulunmaktadır. Örneğin, Tespit, bir izleme bileşeninden belirti sorgularken bu sorgu ilişkili tüm izleme bileşenlerine değil sorgulanan belirtiyi üreten izleme bileşenine yönlendirilmelidir. Bu ihtiyaç ve temel bileşenlerin aynı arayüzü gerçekleyen yalnızca bir temel bileşenle ilişkileri olacak şekilde tasarlanmaları, anahtarlama bileşenlerini gerekli kılmıştır. Bu bileşenler bir temel bileşenin gönderdiği mesajın hangi temel bileşene ulaştırılması gerektiği kararını veren ve ilgili temel bileşene mesajı yönlendiren arakatman bileşenleridir. Bu amaçla mesaj gönderecek temel bileşenin ilgili bağlantı sayısını çoklar. Ayrıca kaynak bilgileri üzerinden hangi mesajın hangi temel bileşenle ilgili olduğu bilgisini tutar. Bu arakatman bileşeninin kullanımı Şekil 8'de gösterilmiştir. Şekilden de anlaşılacağı üzere BAnahtarlama, pA ve pB portları üzerinden BCoklayici'dakine benzer şekilde arayüzleri gerçekler ve kullanır.

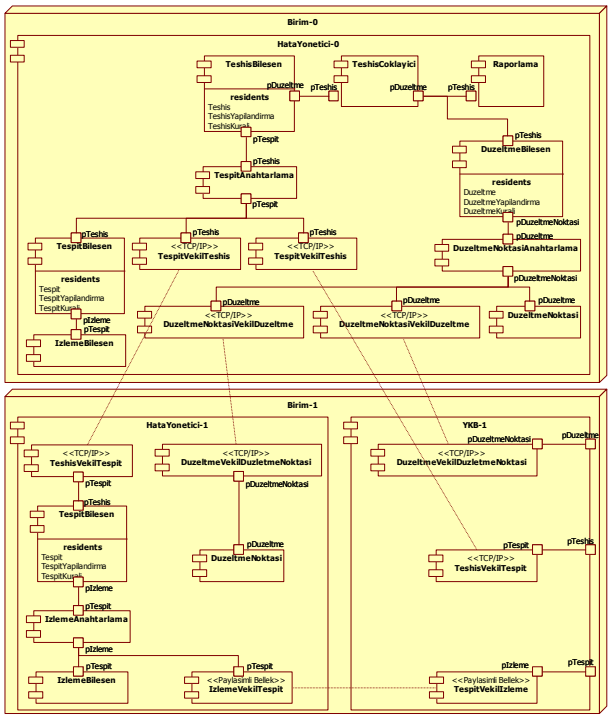


Şekil 8: Anahtarlama Bileşeni Kullanımı

4. Dağıtık Çalışma

Hata yönetimi faaliyetlerinin gerçekleşebilmesi için 2. bölümde açıklanan temel bileşenlerin oluşturulacak sistemde bulunması gerekmektedir. Ancak hangi temel bileşene ait kaç örneğin nasıl konuşlandırılacağı projenin ihtiyaçlarına göre şekillendirilmelidir. Dolayısı ile arakatman bileşenleri temel bileşenlerin konuşlandırma ihtiyaçlarına göre temel bileşenler arası haberleşmeyi destekler nitelikte kullanılır. Bu kullanım sayesinde, temel bileşenler bakış açısı ile ilişkiler sade olmakla birlikte, temel ve arakatman bileşenlerinin kullanımı ile ortaya konulacak ürün ihtiyacın gerektirdiği karmaşıklıkta olabilir. Örneğin, bir temel bileşen, anahtarlama bileşenleri ve çoklayıcı bileşenler sayesinde birden fazla temel bileşen ile ilişki halinde bulunabilir, haberleşme bileşenleri sayesinde başka bir temel bileşen ile farklı haberleşme yöntemleri aracılığı ile etkileşim halinde olabilir.

Şekil 9'da dağıtık bir konuşlandırma örneği verilmiştir. Örnek sistem, Birim-0 ve Birim-1 olmak üzere, iki düğümden oluşmaktadır. Birim-1 üzerinde, hata yönetimi faaliyetleri için bir yazılım konfigürasyon birimi (HataYonetici-1) ile asli görevi hata yönetimi olmayan bir yazılım konfigürasyon birimi (YKB-1) bulunmaktadır. İşlemci ve bellek kullanımının izlenmesi gibi genel izleme faaliyetlerini HataYonetici-1 içerisinde bulunan IzlemeBilesen yürütmektedir. YKB-1'in kontrolündeki altsistemler ve yine YKB-1'in kontrolündeki paylaşımlı kullanılamayan donanımlar ile ilgili izleme faaliyetlerini ise YKB-1 yürütmektedir. Belirtiler, tespit ve teşhis sonuçlarına göre miktar açısından daha fazla olduklarından, belirtilerin maliyetli yöntemlerle başka birimlere aktarılması yerine üretildikleri birim üzerinde işlenmeleri daha akla yatkın bir yöntemdir. Dolayısı ile Birim-1 üzerinde, HataYonetici-1 içerisinde bir TespitBilesen yer almaktadır. YKB-1 tarafından üretilen belirtiler görevler arası haberleşme yöntemi ile YKB-1'e iletilmektedir. TespitBilesen'in uygun izleme bileşenlerinden belirti istemesine olanak sağlamak amacıyla TespitBilesen ile izleme bileşenleri arasında IzlemeAnahtarlama bulunmaktadır. Daha önce açıklandığı üzere anahtarlama bileşeni sayesinde TespitBilesen, birlikte çalıştığı izleme bileşenlerinden soyutlanmıştır. IzlemeAnahtarlama ise <<Paylasimli Bellek>> IzlemeVekilTespit sayesinde ilgili izleme bileşeninin nerede konuşlandırıldığından soyutlanmış durumdadır. Gerek Birim-1 üzerinde üretilen belirtileri işleyen ve HataYonetici içerisinde yer alan TespitBilesen, gerekse YKB-1 ürettikleri tespit sonuçlarını teşhis bileşenine iletirler.



Şekil 9: Dağıtık bir konuşlandırma örneği

TeshisBilesen ve DuzeltmeBilesen sistemin tamamını göz önünde bulundurarak karar verdiklerinden merkezi bir noktada konuşlandırılmaları gerekmektedir. Örnek sistemde ise Birim-0 üzerinde bulunan HataYonetici-0 içerisinde konuşlandırılmışlardır. Birim-0 üzerinde, HataYonetici-0 içerisinde bulunan TespitBilesen tarafından üretilen ve Birim-1'den TCP/IP haberleşmesini kullanan TeshisVekilTespit-TespitVekilTeshis bileşen ikilileri ile ulaşan tespit sonuçları TeshisBilesen tarafından değerlendirilmektedir. TeshisBilesen'in tespit sonucu isteklerini uygun tespit bileşenine yönlendirmek üzere TeshisBilesen ile tespit bileşenleri arasında TespitAnahtarlama yer almaktadır. Diğer taraftan HataYonetici-0 içerisindeki TespitBilesen tek bir izleme bileşeni ile ilişki halinde olduğundan arada bir anahtarlama bileşenine ihtiyaç olmamıştır. TeshisBilesen ürettiği teşhis sonuçlarını düzeltme bileşenine iletir. Ancak örnek sistemde TeshisBilesen'in ilişkili olduğu düzeltme bileşeni aslen bir çoklayıcıdır ve üretilen teşhis sonuçlarını

DuzeltmeBilesen yanında raporlanmak üzere Raporlama bileşenine aktarır.

Daha önce açıklandığı gibi DuzeltmeBilesen, üretilen teşhis sonuçlarına göre düzeltme kararlarının verildiği bileşendir. Asıl düzeltme faaliyetleri ise düzeltme noktası bileşenlerinde yürütülür. Örnek sistemde DuzeltmeNoktasiAnahtarlama, DuzeltmeBilesen'i düzeltme kararlarını hangi düzeltme noktası bileşenine göndereceğinden soyutlamaktadır. TCP/IP haberleşmesini kullanan DuzeltmeNoktasiVekilDuzeltme-DuzeltmeVekilDuzeltmeNoktasi bileşen ikilileri ise düzelteme noktalarının konuşlandırıldıkları yerlerden soyutlamaktadır.

5. Sonuç

Makalede bir hata yönetimi altyapısının tasarımı anlatılmıştır. Tasarım, daha önce sunulan T[5]D kılavuz mimarisine [2] uygun olarak ayrıntılandırılmıştır. Bu tasarım gerçeklenerek projelerde kullanılma aşamasına gelmiştir. İşlevsellik, kullanım kolaylığı, kaynak tüketimi gibi açılardan elde edilen ilk sonuçlar tasarımın ve gerçeklemenin başarılı olduğunu göstermektedir. Diğer taraftan, dağıtık konuşlandırmanın kuralların farklı yapılandırma dosyalarında tutarlı olarak yazılmasında bazı güçlüklere neden olabileceği gözlenmiştir. Bu nedenle kuralların kolay bir şekilde, doğru ve tutarlı olarak üretilmesine yönelik bir araç üzerinde çalışmalar başlamıştır.

T[5]D mimarisi bir ürün hattı yaklaşımı ile ortaya konmuştur. Makaledeki ayrıntılı tasarım da altyapının bir ürün hattı kapsamında kullanılmasına olanak verecek şekilde geliştirilmiştir. Temel ve yardımcı işlevlerin bileşenler olarak modüler hale getirilmesi esnasında bu noktaya dikkat edilmiştir. Ancak geliştirilen bileşenlerden bir projede kullanılacak yazılım konfigürasyon birimlerinin üretilmesi işi henüz geliştiricinin modülleri birleştirici kod yazmasını gerekli kılmaktadır. Bu işi otomatik hale getirilmek amacıyla bir alana özel dil [8] geliştirilmesi de planlanan çalışmalar arasındadır.

T[5]D mimarisinde ismi geçen tahmin işi henüz tasarıma dâhil edilerek gerçeklenmemiştir. Ancak tasarımdaki mevcut arayüzler kullanılarak tasarıma eklenmesinin kolay olacağı öngörülmektedir. Diğer taraftan yine T[5]D mimarisinde yer alan ancak anlatılan tasarımda adreslenmemiş ayırma işi, düzeltmenin bir parçası olarak ele alınmıştır. Gerekirse ayrı ve yeniden kullanılabilir bir bileşen olarak tasarıma dâhil edilecektir.

İşlemci, bellek, işlem izleme gibi işletim sistemine bağımlı izleme işlemleri gelinen aşamada gerçek zamanlı ve gömülü sistemlerde kullanılan vxWorks işletim sistemine özel olarak gerçeklenmiştir. Tasarım her ne kadar görev kritik ve gömülü sistemler için geliştirilmiş olsa da masaüstü bilgisayarlar ile de etkileşimli çalışma öngörülmektedir. Dolayısı ile Windows ve Linux tabanlı işletim sistemlerine özel izleme de planlanan çalışmalar arasındadır. Windows işletim sistemine özel izleme ODTÜ Bilgisayar Mühendisliği Bölümü bünyesinde ortaya konulan bir yüksek lisans tezi [9] kapsamında geliştirilen çerçeve, Linux işletim sistemine özel izleme ise Bilkent Üniversitesi Bilgisayar Mühendisliği Bölümü bünyesinde bir bitirme projesinde [10] geliştirilen altyapı kullanılarak sağlanacaktır.

Bundan sonraki aşamada, altyapıyı kullanan projelerden gelecek geri beslemeler ile tasarıma yeni eklemeler ya da gerçeklemede değişiklikler olabilecektir. Ayrıca ODTÜ Bilgisayar Mühendisliği Bölümü ile planlanmakta olan bir ortak çalışma ile tespit, teşhis ve düzeltme kurallarının geliştirilmesi, ayrıca düzeltme faaliyetleri ile ilgili olarak bir denetim noktası (checkpointing) yaklaşımının belirlenmesi öngörülmektedir. Bu çalışmanın çıktıları da mevcut tasarıma dâhil edilecektir.

6. Teşekkür

Bu çalışmanın temellerini oluşturan T[5]D mimarisinin oluşturulmasında emeği geçen ekibe, makalede anlatılan tasarımın gerçeklenmesine destek veren Burak Kekeç ve Alper Aygar'a, ayrıca değerli fikirlerini bizlerle paylaşarak bu çalışmanın hayata geçmesini sağlayan tüm çalışma arkadaşlarımıza teşekkür ederiz

7. Kaynakça

[1] L. Alkışlar, "ASELSAN'da Yazılım Mimarileri Uygulamaları", Ulusal Yazılım Mimarisi Konferansı, İstanbul, 2006.

[2] M. Tekkalmaz, Ö. Kaya, M. Dursun, T. Sarı Tekkalmaz, A. Doğru, "Görev Kritik ve Gömülü Sistemler için Hata Yönetimi Kılavuz Mimarisi: T[5]D", 2. Ulusal Yazılım Mimarisi Konferansı, İzmir, 2008.

[3] K. Pohl, G. Böckle, ve F. v. d. Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, Berlin Heidelberg New York, 2005.

[4] International Organization for Standardization (ISO), "Information technology – Syntactic metalanguage – Extended BNF", *International Organization for Standardization*, ISO/IEC 14977 : 1996, 15 Aralık 1996.

[5] D. Grune, H. Bal, C. Jacobs, K. Langendoen, *Modern Compiler Design*. John Wiley & Sons, Ltd., 2000.

[6] Robert C. Martin. "The Open-Closed Principle," *C++ Report*, Ocak 1996.

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissidies, *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] M. Mernik, J. Heering, A. M. Sloane. "When and how to develop domain-specific languages". *ACM Computing Surveys*, 37(4):316–344, 2005.

[9] K. Kuz, "Design and Implementation of a Fault Monitoring Framework", Yüksek Lisans tezi, Orta Doğu Teknik Üniversitesi, Ankara, Türkiye, 2009.

[10] B. Yıldız, S. Bacanlı, E. Varol, O. Kışlal, E. Uğur, "Adaptable Rule-Based Fault Detection System", Bitirme Projesi, Bilkent Üniversitesi Bilgisayar Mühendisliği Bölümü, Ankara, Türkiye, 2009. Çevrimiçi Proje Sayfası: http://www.ug.bcc.bilkent.edu.tr/~kislal/ (Arşivlenen daimi adres: http://www.webcitation.org/5gvsdoTxS)