Genel Amaçlı Haberleşme Arayüzü Simülatör Yazılımı

Mustafa GEVHER [1] Metin TEKKALMAZ [2] Hakan YAMANYAR [3] Ömer BİLGİN[4]

[1, 2, 3, 4] Radar Elektronik Harp ve İstihbarat Sistemleri (REHİS) Grubu, ASELSAN A.Ş. Ankara

{[1]mgevher, [2]tekkalmaz, [3]hyamanyar, [4]omerbilgin}@aselsan.com.tr

Özet. Farklı projelerde geliştirilen yazılımların diğer çevre birimleri ile ilgili olan arayüzlerinin testlerinde kullanılmak üzere geliştirilmekte olan simülatör yazılımlarının ortak bir altyapı ile hazırlanması amacı ile Genel Amaçlı Simülatör yazılımı (GAS) geliştirilmiştir. Bu makalede farklı arayüz ve protokolleri (TCP[1]/UDP IP [2], ARINC 429 [3], CAN Bus [4], Ayrık hat, RS 232 [5]/RS 422 [6], MIL-STD-1553B [7]) ortak bir mesaj tanımı şablonu ile tanımlayıp, dinamik olarak güncellenebilen kullanıcı arayüzü geliştirilmesini sağlayan Genel Amaçlı Simülatör Yazılımının tasarımı ve kullanımı değerlendirilmektedir.

Abstract. Generic Simulator was developed in order to provide a common framework for generating simulation software which are used to test communication interfaces used by the software units developed for various projects. This paper evaluates the design and usage of Generic Simulator Software which provides a framework that defines common message definition templates for various interfaces and protocols (TCP/UDP IP, ARINC 429, CAN Bus, Discrete line, RS 232/422, MIL-STD-1553B) and also enables the user to define and update the graphical user interface (GUI) dynamically.

Anahtar Kelimeler*:* simülatör, protokol, yazılım test ve doğrulama

1 Giriş

Günümüzde yazılımlar yalnızca kullanıcı ile etkileşim halindeki bağımsız yapılar olmakla kalmayıp, donanım ve diğer yazılımlar ile iletişim içerisinde çoğunlukla karmaşık sistemlerin parçalarını oluştururlar. Web ya da masaüstü uygulamaları genellikle TCP veya UDP protokolleri üzerinde çalışan FTP ve HTTP gibi yaygın uygulama katmanı haberleşme protokollerini kullanır. Diğer taraftan gömülü sistem yazılımları, bunlar gibi yaygın haberleşme protokollerinin yanı sıra MIL-STD-1553B, RS-232, RS-422, ARINC-429, CAN Bus ve dijital/analog girdi/çıktı hatları gibi daha az yaygın olan ve kullanım alanına göre özelleşmiş haberleşme yöntemlerine de ihtiyaç duyabilmektedir.

Yazılım testlerinde kullanılan arayüz simülatörlerini hazırlamak, yazılımın arayüz sayısına ve arayüzlerin karmaşıklığına göre uzun süren bir faaliyet olabilmektedir. Yukarıda bahsi geçen farklı haberleşme protokollerinin ele alınması, simülatör kullanıcı arayüzünün hazırlanması, alınan ve gönderilen verinin bu kullanıcı arayüzü üzerinden sunulması farklı arayüz simülatörlerinde tekrarlanan ancak simülatörün geliştirilmesinde kayda değer yer tutan faaliyetlerdir.

Genel Amaçlı Simülatör (GAS), arayüz simülatörlerinin hazırlanması işini hızlandırma hedefi ile ortaya çıkan bir çalışma olmuştur. Bu amaçla GAS kapsamında aşağıdaki hedefler belirlenmiştir:

- Simülatör geliştiricisi, alt tarafta kullanılan haberleşme protokollerinden mümkün olduğunca soyutlanmalıdır,
- Kullanıcı arayüzünün hazırlanması için standart bir yöntem kullanılmalıdır,
- Haberleşme esnasında alınan/gönderilen verinin kullanıcı arayüzü ile ilişkilendirilmesi işleri kolay ve ortak bir yöntemle yapılabilmelidir,
- Simülatörü hazırlanan yazılımın/birimin davranışını tanımlamak için bir yöntem sunulmalıdır.

Bu makalede GAS, gerek tasarımı gerekse kullanımı açılarından tanıtılmaktadır. Bölüm 2'de, GAS yetenekleri, GAS'a ait genel yazılım mimarisi, temel sınıflar arası ilişkiler, kullanıcı arayüzünün hazırlanması için sunulan altyapı yukarıda bahsi geçen hedefler gözetilerek verilmektedir. Bölüm 3'te GAS kullanıcı deneyimleri, gelişmeye açık olası konular ile birlikte sunulmaktadır. Son olarak genel bir değerlendirme sunularak makale tamamlanmaktadır.

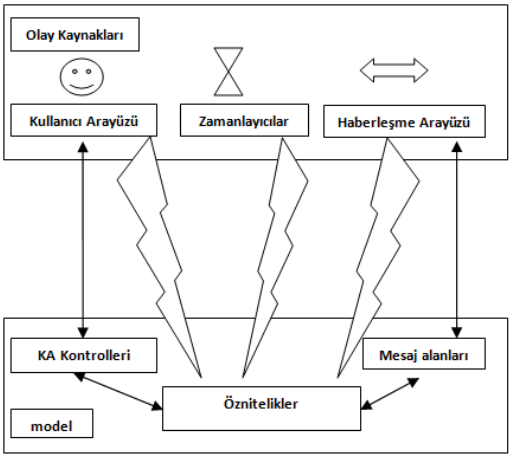
2 Simülatör Yazılımı Tasarımı

Bu bölümde Genel Amaçlı Simülatörü oluşturan temel yapıtaşları tanımlanmaktadır. Simülatör yazılımının hazırlanması için ayrıca GAS Konfigürasyon Tanımlama Yazılımı (KTY) geliştirilmiştir. Bu araç ile kullanıcı arayüzü (KA) ve haberleşme arayüzü arasındaki ilişkiler tanımlanabilir. Kullanıcının ihtiyaç duyması halinde değerlendirme ve kontrol kodlarını yazması için arayüz sağlar. Bunun yanı sıra SOAP [8], ASN1 [9], CORBA [10] gibi standartların kullanılması için gereken altyapıların simülatöre eklenmesi için arayüz de sağlamaktadır.

2.1 Kullanıcı-Haberleşme Arayüzlerinin Senkronizasyonu

Yazılımların testlerinde kullanılan simülatörlerin temel olarak gerçekleştirmesi gereken işlevlerden birisi kullanıcı arayüzü (KA) giriş kontrollerinin gönderilecek mesajlara ait alanları güncelleyebilmesi ve gelen mesajların içerisinde bulunan alanlardaki bilgilerin KA gösterim kontrollerini güncelleyebilmesidir. Bu amaçla birbiri ile etkileşecek olan KA ve haberleşme mesaj alanları öznitelikler (attribute) aracılığıyla birbirine bağlanmıştır (**Şekil 1**). Güncellenen mesaj alanlarına/(KA kontrollerine) bağlı oldukları KA kontrollerinin/(mesaj alanlarının) değerleri otomatik olarak güncellenmesinin yanı sıra, yazılım içinde ihtiyaç duyulması halinde kullanıcının "business logic" olarak tanımlanan, yeni olaylar ve güncellenen değerlere

göre karar verilmesi gereken durumları yönetebilmesi için C# dili ile kodlama altyapısı KTY tarafından sağlanmıştır.

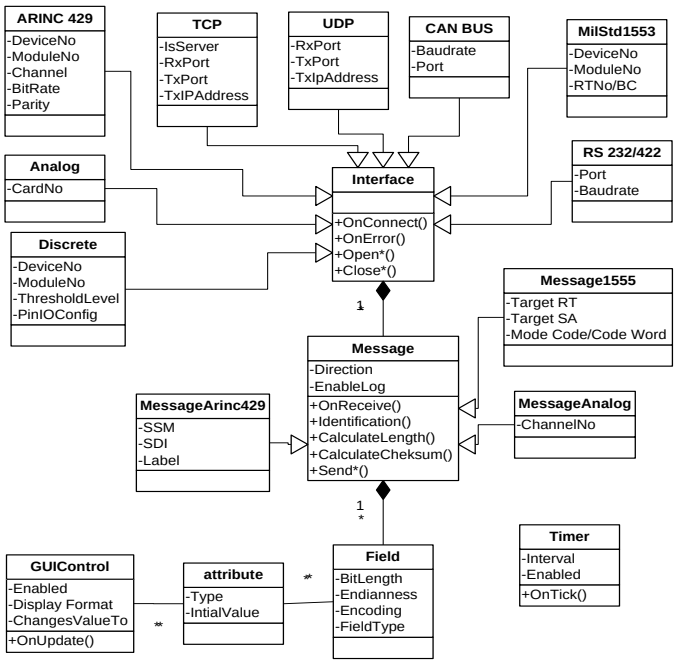


Şekil 1 Genel Amaçlı Simülatör İşleyişi

GAS yazılımı temel olarak 3 olay kaynağını kullanır:

1. Kullanıcı Arayüzü: Kullanıcının (form üzerindeki) yaptığı değişiklikler ya da gelen mesajlara ait alanlar, bağlı bulunduğu özniteliği değiştirir. Buna bağlı diğer KA kontrolleri ya da mesaj alanları var ise bunlar da otomatik değişir. Bununla beraber "OnUpdate" fonksiyonu çağrılarak güncellenen öznitelik ile beraber yapılması gereken başka işlemler de tanımlanabilir.
2. Zamanlayıcılar: Zamanlayıcılar tanımlandığı zaman aralığı sonunda "OnTick()" fonksiyonunu çağırır. Bu fonksiyon içerisinde istenilen işlevi yerine getirmek üzere kullanıcı kodlama yapabilir.
3. Haberleşme arayüzü: GAS aracılığıyla geliştirilmekte olan simülatör yazılımının kullandığı haberleşme arayüzü bağlantıya dayalı (connection-oriented) bir protokol ise, bağlantı sağlandığında yapılacak işlemlerin tanımlanması "OnConnect" fonksiyonu içerisinde yapılır. Bunun yanında "OnError" fonksiyonu ile arayüzde oluşacak hatalar için gereken işlemlerin yapılmasına olanak sağlanır. Ayrıca, "OnReceive" fonksiyonu ile arayüz kapsamında gelen mesaj tipinde tanımlanmış mesajların alınması durumunda yapılması gereken işlemler tanımlanabilir. "OnReceive" fonksiyonu ile mesaj alanına ait KA kontrolleri varsa bunlar da otomatik olarak güncellenip, kontrolun "OnUpdate" fonksiyonu çağrılır.

GAS yazılımının çekirdeğini oluşturan temel sınıfların ilişkileri ve içerikleri sadeleştirilmiş olarak **Şekil 2** de gösterilmiştir.



Şekil 2 GAS Temel Tasarımı

2.2 Kullanıcı Arayüzünün Tanımlanması

GAS kullanıcı arayüzünün hazırlanması Microsoft firması tarafından geliştirilmiş olan XAML (Extensible Application Markup Language[11]) standardına göre hazırlanan KA tanım dosyası ile sağlanmaktadır. GAS, kullanıcı tarafından yazılan XAML dosyasını işler, buna göre kullanıcı arayüzünü otomatik oluşturur. XAML içinde tanımlı KA kontrolleri özniteliklere bağlanarak olay fonksiyonları içinde KA kontrollerinin girdi/çıktı ve gerekiyorsa güncelleme işlemleri KTY aracılığıyla tanımlanır.

2.3 Haberleşme Arayüzlerinin Ortak Şablonda Tanımlanması

Test edilecek yazılım ve birimler çok farklı arayüz ve protokolleri içerebilmektedir. Bu amaçla KTY kullanıcı arayüzü kullanarak haberleşme arayüz tanımının ortak bir şablon içerisinde tanımlanması, arayüze özel bilgilerin de bu ekranları kullanarak ayarlanması sağlanmıştır. Desteklenen protokol ve haberleşme/donanım arayüzleri şunlardır: RS-232/422, TCP IP, UDP IP, MIL-STD-1553B, ARINC-429, CAN Bus, Analog Giriş/Çıkış Sinyalleri, Ayrık Hat (Sayısal) Giriş/Çıkış Sinyalleri.

Kullanıcı, ihtiyacı olan haberleşme/protokol/sinyal arayüzünü ilgili arayüz ayarlarını da girerek tanımlar. Daha sonra arayüz içindeki gelen ve giden mesajları arayüz tanımının altına ekler. Mesaj ayarları girildikten sonra her mesaj için ilgili mesajı oluşturan mesaj alanlarını tanımlar. Tanımlanan alanları özniteliklere bağlar. Böylece gönderilecek mesaj alanına bağlı özniteliklerin değiştirilmesi sonrasında

"MesajYolla" fonksiyonunun çağrılmasıyla özniteliklere bağlı alanlar güncellenerek gönderilir. Benzer şekilde dış arayüzlerden gelen mesajları oluşturan alanlar da bağlı bulundukları özniteliklerin değerlerini günceller.

Haberleşme arayüzü/protokolü/sinyali ne olursa olsun kullanıcı her arayüz için ortak olan, "MesajYolla(SendMessage)" fonksiyonunu ve "MesajAlındı(OnReceive)" fonksiyonunu standart olarak kullanır. Dolayısıyla, haberleşme arayüzlerinin protokol seviyesindeki yazımının tekrarı engellenmiş ve kullanıcının yapacağı işlem arayüz tanımı ve ilişkilerin kurulması şekline indirgenmiş olur.

2.4 "Business Logic" Tanımlanması

GAS yazılımı ile oluşturulan simülatörlerin genel kullanımında, simülatörden sadece ekran bilgileri ile mesajlaşma çoğu zaman yetersiz kalmaktadır. Bu durum karşısında kullanıcının simülatörde gerçekleşen olaylarda değerlendirme, karar verme ve işlem yapması gerekir. Bu kısımlar için kullanıcının C# dili ile kodlama yapabilmesi için altyapı sağlanmıştır.

2.5 Kullanıcıya Ait Bileşen Sınıf ve Fonksiyonların Eklenmesi

Daha önce yazılmış olan bileşen (*.dll), C# sınıfı ve fonksiyonları da istenirse simülatör yazılımına eklenip çağrılabilmektedir. Böylece hazır bileşenlerin tekrar kullanımı da sağlanmış olmaktadır. Bu amaçla KTY arayüzünde bileşen, sınıf ve fonksiyon ekleme arayüzü sağlanmıştır.

2.6 SOAP,ASN.1 CORBA Desteği

GAS yazılımı donanım arayüzleri, haberleşme protokollerinin yanı sıra objelere ait verilerin belirli bir veri yapısı standardı ile ağ içerisinden aktarımını sağlayan SOAP, CORBA, ASN.1 gibi standartları desteklemektedir.

GAS ile kullanılabilen SOAP örneği **Şekil 3**'de, ASN.1 örneği **Şekil 4**'te, CORBA örneği **Şekil 5**'te gösterilmektedir.

```
// Define class instance to serialize
Person aPerson = new Person();
// Serialize instance into a Byte array
Byte[] aByteArray = Soap.Serialize(aPerson);
// Deserialize the Byte Array into a class instance
Person NewPerson = (Person)Soap.Deserialize(aByteArray);
```

Şekil 3 SOAP Örnek Kulanımı

```
// DataReqMessage, DataMessages, UNIT_Msg ".asn" ve ".cfg" dosyaları //
kullanılarak "asn1c.exe" derleyicisi tarafından üretilmiştir.

int NumberToSent = 2;
DataReqMessage dataReqMessage = new DataReqMessage(NumberToSent);
DataMessages dataMessages = new DataMessages(DataMessages._DATAREQ,
dataReqMessage);
UNIT_Msg gonderilecekMsg = new UNIT_Msg(UNIT_Msg._DATAMSG, dataMessages);
Asn1BerEncodeBuffer encodeBuffer = new Asn1BerEncodeBuffer();
gonderilecekMsg.Encode(encodeBuffer);
Byte[] aByteArray = new Byte[encodeBuffer.MsgLength];
aByteArray = encodeBuffer.MsgCopy;

//Decode asn1, BER encoded Buffer
Asn1BerDecodeBuffer decodeBuffer = new Asn1BerDecodeBuffer(aByteArray);
UNIT_Msg alinanMsg = new UNIT_Msg();
alinanMsg.Decode(decodeBuffer);
if (alinanMsg.ChoiceID == UNIT_Msg._DATAMSG)
{
   DataMessages alinanDataMessages = (DataMessages)alinanMsg.GetElement();
   if (alinanDataMessages.ChoiceID == DataMessages._DATAREQ)
   {
      DataReqMessage RxDataReqMessage
                        =(DataReqMessage)alinanDataMessages.GetElement();
      int NumberReceived=uint.Parse(RxDataReqMessage.msgID.ToString());
   }
}
```

Şekil 4 ASN.1 Örnek Kullanımı

```
//Server initialization
void InitCorba() {
   //parameters:
   //1234: server port
   //aGreeting: instance of concrete class
   //"Greetings": name of the inteface
   //"remote.ior": name of the ior file which contains Corba server parameters
   Corba.InitCorbaServer(12345,aGreeting,"Greetings","remote.ior");
}
//Client initialization
void InitCorba() {
   //parameters:
   //aGreeting: instance of interface
   //Greetings: inteface type name
   //"remote.ior": name of the ior file which contains Corba server parameters.
   aGreeting=(Greetings)Corba.ConnectCorbaServer("remote.ior",typeof(Greetings));
}
```

Şekil 5 Örnek CORBA Kullanımı

3 Kullanıcı Değerlendirmesi

GAS yazılımı, sunduğu modüler tasarımı ile kullanıcıya ihtiyacı olan simülatör yazılımlarını daha kısa sürede hazırlama imkanı verir ve işgücünden kazanç sağlar. GAS yazılımı, önceden hazırlanan derlenmiş bileşenleri (*.dll) kullanabilme ve derlenmeye dahil olacak C# sınıflarının gerekirse sahada dahi güncellenebilmesine izin verme yeteneği ile, benzer görevlere sahip simülatör yazılımlarının çoğaltılmasını kolaylaştırır.

Test edilen yazılım ile gerçekleşen haberleşme sırasında alınan ve gönderilen mesajlara test amaçlı kullanılan yapay hatalar (error injection) eklenebilir. GAS yazılımı mesajlara, sağlama toplamı (checksum) hatası, artık veri (redundant bytes) hatası, kayıp veri (missing bytes) hatası ve mesaj engelleme (block message) hatası ekleyebilmektedir. Bu özellik test senaryosunda çeşitlilik oluşturulabilmesine imkân sağlar.

Kullanıcı arayüzü tasarımı sırasında, her bir kullanıcı arayüz kontrolü manuel olarak yaratılması gereken bir özniteliğe bağlanmalıdır. Her ne kadar GAS yazılımı ile bir simülatör yazılımının geliştirilmesi, manuel simülatör yazılımı geliştirmeye

göre daha hızlı olsa da, her bir kullanıcı arayüz kontrolü için manuel olarak öznitelik yaratılması simülatör yazılımı geliştirme süresini uzatan en büyük faktördür. GAS yazılımına eklenecek bir yetenek ile simülatör geliştirme işçilik süresinin daha da düşürülmesi sağlanabilir.

4 Sonuç

Bu makalede yazılım testlerinde kullanılan arayüz simülatörlerini esnek bir altyapı sunarak, kullanıcı ve haberleşme arayüzlerini mümkün olduğunca konfigürasyon dosyaları ile tanımlamaya imkan veren Genel Amaçlı Simülatör yazılımın tanıtımı yapılmıştır. Geliştirilen GAS yazılımı çeşitli projelerde kullanılmış ve yazılımın haberleşme arayüz testlerinde kullanımak üzere ihtiyaç duyulan simülatörlerin geliştirme süresini önemli ölçüde kısalttığı gözlemlenmiştir. Test yazılımı geliştiricisinin, kullanması gereken arayüz ve protokol detaylarına boğulmadan genel bir şablonu takip etmesini sağlayarak gereken simülatörün geliştirilmesi sürecini kolaylaştırmıştır. Gelişen teknolojiye uygun olarak geliştirilen yeni donanıma göre altyapının güncellenmesi ile kullanım alanının genişletilebileceği değerlendirilmektedir.

5 Kaynakça

1. Vinton G. Cerf; Robert E. Kahn (May 1974). "A Protocol for Packet Network Intercommunication"
2. Postel, J., "User Datagram Protocol", RFC 768, August 1980
3. ARINC Specification 429P1-15, September 1, 1995
4. BOSCH. CAN Specification. Version 2.0. 1991, Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart
5. EIA standard RS-232-C: Interface between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange. Washington: Electronic Industries Association. Engineering Dept. 1969.
6. TIA/EIA STANDARD, Electrical Characteristics of Balanced Voltage Digital Interface Circuits, TIA/EIA-422-B, May 1994
7. 1553:US Department Of Defense, 1978, MIL-STD-1553B Aircraft Internal Time Division Command/Response Multiplex Data Bus.
8. https://www.w3.org/TR/soap/
9. http://www.itu.int/itu-t/recommendations/rec.aspx?rec=x.680
10. Object Management Group, The Common Object Request Broker: Architecture and Specification (Draft). Revision 1.1, 1991
11. https://msdn.microsoft.com/en-us/windows/uwp/xaml-platform/xaml-overview
12. https://github.com/thinkpixellab/kaxaml