Dağıtık Sistemler İçin Haberleşme Otomasyon Ara Katmanı: ULAK

Burcu Karasoy[1], Soner Çınar[1]

[1]Yazılım Mühendisliği Müdürlüğü, Mühendislik Dir., SST Grup Bşk.
ASELSAN A.Ş.

{bkarasoy, scinar}@aselsan.com.tr

Özet. Geliştirilen sistemlerin kapsam ve yetenekleri arttıkça, çok sayıda yazılım ve bileşenin birbiriyle haberleşme ihtiyacı ortaya çıkmıştır. Haberleşme arayüzlerini her farklı yazılım ya da bileşende yeniden kodlamak, arayüz değişikliklerinde değişen kısımları elle güncellemek hem zaman kaybına, hem de hataya yatkın yazılım parçaları geliştirilmesine neden olmaktadır. Diğer yandan gömülü yazılımlarda haberleşmeden kaynaklanan hatalar, yazılım güvenliği açısından oldukça kritiktir. Bu tür sorunlara çözüm getirmek amacıyla haberleşme otomasyon ara katmanı ULAK geliştirilmiştir. ULAK sayesinde tüm haberleşme altyapıları (çözümleme ve formatlama) otomatik olarak oluşturulabilmekte, farklı platformlardaki yazılımlar karşılıklı olarak haberleşebilmektedir (endian sorunu).

Anahtar Kelimeler: Yazılım Mimarileri, Haberleşme Ara Katmanı, Çapraz Platform Haberleşmesi

1 Giriş

Dağıtık sistem, farklı bilgisayarlardaki yazılım bileşenleri arasındaki haberleşme ve koordinasyonun sadece mesajlaşma yoluyla sağlanabildiği ağ olarak tanımlanabilir. Haberleşme, mesajların sistemdeki bütün yazılım bileşenlerinin tanıdığı bir formatta gönderimi ve karşı tarafın bu mesajları çözümlemesi ile gerçekleşir. Mesajlar elle yazılımcı tarafından kodlanabileceği gibi bu iş için oluşturulmuş hazır bir ara katman yazılımı da kullanılabilir.

Elle kodlama en basit çözüm olarak görünse de haberleşen birim sayısı, arayüz çeşitliliği ve değişiklikleri artıkça sistemin sağlıklı bir haberleşme kurabilmesini sağlamak kritik bir sorun halini alır. Arayüzlerin tek tek elle kodlanması sistemi hataya yatkın bir hale getirirken ciddi bir iş gücü kaybına da sebep olmaktadır. Farklı platformlardaki yazılımların haberleşme gereksinimlerinin de karşılanması gerekliliği haberleşme kodlarını iyice karmaşıklaştırmaktadır.

Günümüzde dağıtık sistemlerdeki haberleşme sorununu çözmek için oluşturulmuş pek çok hazır ara katman yazılımı da bulunmaktadır. Ancak bu yazılımların kullanımında pek çok sıkıntı ile karşılaşılabilmektedir. Bu araçlar çok geniş bir sorun küme-

sine çözüm olarak oluşturuldukları için pek çok karmaşık yapıyı da beraberlerinde getirebilmektedirler. Karmaşıklık ise sistemleri hataya daha açık bir hale getirir. Özellikle az sayıdaki sistemlerden oluşan ve hatasız çalışmanın çok önemli olduğu silah sistemlerinde bu teknolojiler avantajdan çok dezavantaj getirebilmektedir.

ULAK, özellikle az sayıda yazılımdan oluşan gerçek zamanlı gömülü sistemlerdeki haberleşmeyi sağlamak için oluşturulmuş bir ara katmandır. Haberleşmeyi, gecikmeyi minimuma indirerek kısıtlı kaynak kullanımı ile gerçekleştirmek, çoklu görev yürütümünün olduğu yazılımlarda görev bölünmesine karşı önlem almak ULAK'ın öncelikleri arasındadır. Bildirinin bir sonraki bölümünde dağıtım ara katman teknolojilerinden kısaca bahsedilecek, sonraki kısımlarda ise ULAK mimarisi tanıtılacaktır.

2 Dağıtım Ara Katmanı Teknolojileri

Dağıtım ara katmanı teknolojileri kabaca üç gruba ayrılabilir: istemci-sunucu (client - server), mesaj yönelimli (message oriented) ve yayınla-abone ol (publish - subscribe) [1].

İstemci - Sunucu

İstemci-sunucu tasarımını benimseyen ağlar veriyi işleyen ya da depolayan sunucu ile veriyi isteyen ve kullanan istemciyi birbirine bağlar. Çoğu istemci-sunucu ara katmanı uzaktaki objelerin birbirlerinden yerel makinadaymış gibi fonksiyon çağrısı yapabilmesini (Uzak Metot Çağrısı, "*Remote Method Invocation – RMI*") sağlayan bir Uygulama Programlama Arayüzü (UPA) sunar. İstemci-sunucu tasarımları merkezleştirilmiş bilginin bulunduğu veritabanları, hareket işlem sistemleri ve merkezi dosya sistemi gibi sistemlerde genellikle başarılı bir şekilde çalışır. Bu türdeki başarılı tasarımlara örnek olarak CORBA, DCOM, HTTP ve Enterprise JavaBeans (EJB) gösterilebilir [2, 9]. Veri akışının çoktan bire (many-to-one) olduğu durumlarda istemci-sunucu tasarımı verimli bir şekilde çalışır. Birden fazla birimin veri ürettiği durumlarda ise istemci-sunucu mimarisi verinin dağıtım amacıyla sunucuya gönderilmesini gerektirir. İstemciler arasında bu türden dolaylı bir haberleşme, gecikmeye sebep olacağından, özellikle gerçek zamanlı sistemlerde istenmeyen bir durumdur.

Mesaj Yönelimli

Mesaj yönelimli mimarilerde haberleşme mesajın doğrudan iletilmesiyle ya da güvenilir kuyruk yapılarının kullanımı ile sağlanır. Senkron ve asenkron etkileşim tiplerinin her ikisini de desteklemesine rağmen bu mimariler birincil olarak asenkron iletişim için oluşturulmuştur[3]. Kuyruk kullanılarak asenkron iletişim, gönderilen mesajın alıcı tarafından alınana kadar kuyrukta tutulması ile gerçekleştirilir. Bu mimaride her bir istemcinin diğer istemcilerle mesaj alışverişi kolaylıkla sağlanır[4]. Bu sebeple istemcilerin haberleşmesi gereken mimariler için istemci-sunucu mimarilerine göre daha uygundur. IBM MQSeries [10] ve Sun Java System Message Queue [11] bu kategorideki ürünler arasındadır.

Yayınla-Abone Ol
Bu mimari kısaca haberleşen birimlerin ihtiyaçları olan verilere abone olması ve ürettikleri verileri yayınlamaları esasına dayanır. Mesajlar herhangi bir ara sunucuya ihtiyaç olmadan doğrudan haberleşme birimleri arasında gidip gelebilir [2]. Yayıncılar ve aboneler birbirlerinin sayısını, konumunu, canlı olup olmadığını bilmek zorunda değildir [5]. Bu sistemler genelde büyük miktarda zaman-kritik veriyi dağıtmak için kullanılırlar. Object Management Group'un yayınladığı Data Distribution Service (DDS) standardı gömülü sistemler için oluşturulmuş ilk açık uluslararası yayınla-abone ol mimarisi standardıdır [6]. Diğer yaygın yayınla-abone ol ara katmanlarına örnek olarak ZeroMQ [7] ve ORTE [8] verilebilir.

3 ULAK Ara Katmanına Genel Bir Bakış

ULAK, aynı ya da farklı platformlardaki yazılım ve bileşenlerin görev bölünmelerinden etkilenmeyecek şekilde haberleşmesini sağlayan noktadan noktaya haberleşme ara katman mimarisidir. ULAK-Builder yazılımı sayesinde ara katman otomatik olarak oluşturulur ve arayüzlerin tekrar tekrar elle kodlanmasına gerek kalmaz. Bu yazılım girdi olarak haberleşme arayüzlerini fonksiyon imzası olarak alır ve tüm haberleşme alt yapılarını (çözümleme ve formatlama) otomatik olarak oluşturur. Bu yönüyle ULAK teknolojisi, dönüşüm seviyesinde yeniden kullanıma (transformation-based reuse) olanak sağlar.

ULAK mimarisi az sayıda sistemden oluşan dağıtık sistemler arasındaki haberleşmeyi ek bir sunucu vs. gerektirmeksizin, ekstra zaman kaybı olmadan uzak metot çağrısı ile sağlar. ULAK, uzak metot çağrısına olanak vermesi sebebiyle en yaygın ara katmanlardan biri olan CORBA'ya benzetilebilir. Ancak CORBA'nın fazla miktarda veri kopyalanması ve dönüşüm yapılması gibi sebeplerle sistemde yüksek gecikmeler oluşturabildiği bilinmektedir [13,14]. ULAK, gerçek zamanlı sistemlerin ihtiyaçları ve kısıtları göz önünde bulundurularak oluşturulmuştur. Haberleşme kaynaklı gecikmeleri minimum düzeyde tutmak ULAK'ın önceliklerinden biridir. Gerçek zamanlı sistemlerde uzak metot çağrısına imkan vermesi bu ara katmanın en önemli özelliklerindendir.

3.1 ULAK Ara Katmanının Yetenekleri

ULAK teknolojisi ile hazırlanmış bir yazılım ya da bileşen aşağıdaki yetenekleri de kazanmış olur:

- Yazılım seviyesinde tüm haberleşmeler fonksiyon çağrısından ibarettir.
- Her yazılım, kendi yazılımı içerisinde yer alan bir sınıfın fonksiyonuna erişir gibi harici yazılımın fonksiyonuna erişebilir.
- Fonksiyonun tanımına göre, parametreli, parametresiz, dönüş değeri olan ya da olmayan fonksiyonlara erişilebilir. Veri ve komut gönderimi-alımı sağlanır.
- Yazılımların çalıştığı platform uyumu (big endian / little endian) otomatik sağlanır.

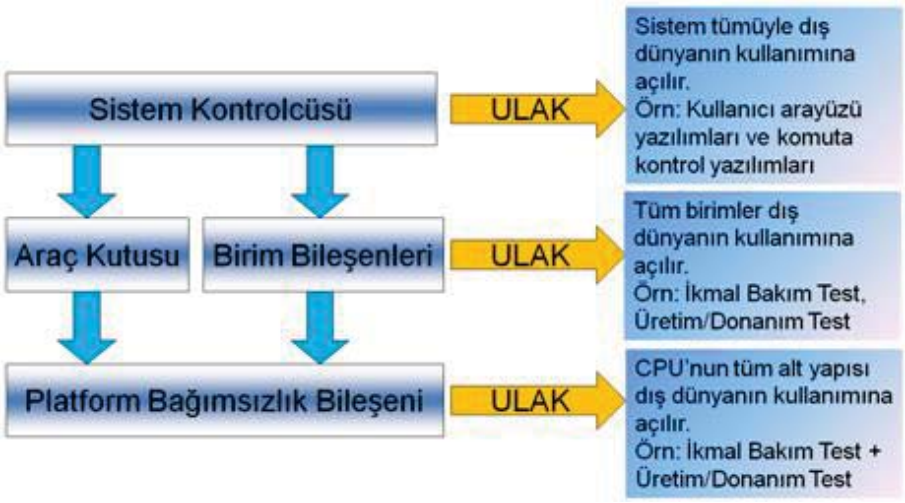
- ULAK motorunun kullanacağı haberleşme protokolü seçilebilir. TCP/IP, UDP, seri kanal gibi protokoller ile çalışılabilir.
- Çoklu görev yürütümünün olduğu yazılımlarda görev bölünmesine karşı korumalıdır.
- Fonksiyon çağrıları, isteğe göre, karşı tarafta işlenene kadar bekletilir ya da bekletilmez (senkron-asenkron çalışma).
- Bağlantı durumu periyodik olarak kalp atışı (heartbeat) yöntemi ile ULAK tarafından otomatik kontrol edilir.
- Yazılımlar, bağlantının durumunu sorgulayabilir ya da geri dönüş fonksiyonu alt yapısı ile değişikliklerden asenkron haberdar olabilir.
- Haberleşme kanalından akan tüm mesajların formatlarını tarifleyen arayüz dokümanı otomatik olarak oluşturulur.
- Daha önceden ULAK aracıyla hazırlanmış sürücülerde yapılan arayüz değişiklikleri, eklemeleri, çıkartmaları yine bu araç yardımıyla kolaylıkla yapılır.
- Haberleşen birimlerin aynı CPU'da olması durumunda haberleşmenin haberleşme kanalı yerine doğrudan bağlantı ile gerçekleşmesi sağlanabilir. Bu durumda haberleşmenin hangi yolla gerçekleştirileceği (haberleşme kanalı ya da doğrudan) çalışma zamanında belirlenebilir.

3.2 Kullanım Alanları

Çok farklı amaçlar için kullanılabilen ULAK'ın temel olarak kullanım alanlarını aşağıdaki şekilde listelemek mümkündür.

- İki yazılım arasında haberleşme ara katman rolü
- Bileşenin harici yazılımlar tarafından kullanımı
- Yazılımın, bileşenler bazında farklı işlemcilere dağıtılması
- Yazılımın/Bileşenin kara kutu testi
- Yazılımın/Bileşenin Windows Kontrol Yazılımı
- Yazılımın/Bileşenin Windows Kontrol Yazılımı ile testi
- Dokümantasyon

Şekil 1'deki örnek yapıya benzer bir mimariye sahip bir sistem yazılımı, ULAK sayesinde üç seviyede yeteneklerini genel kullanım amaçlı dış dünyaya açabilir. Bu şekilde sistem kontrolcüsü kullanıcı arayüzü ve komuta kontrol yazılımları ile haberleşebilir. Araç kutusunun testi, birim bileşenlerinin ve alt donanımın ikmal bakım ve üretim/donanım testleri harici olarak gerçekleştirilebilir.



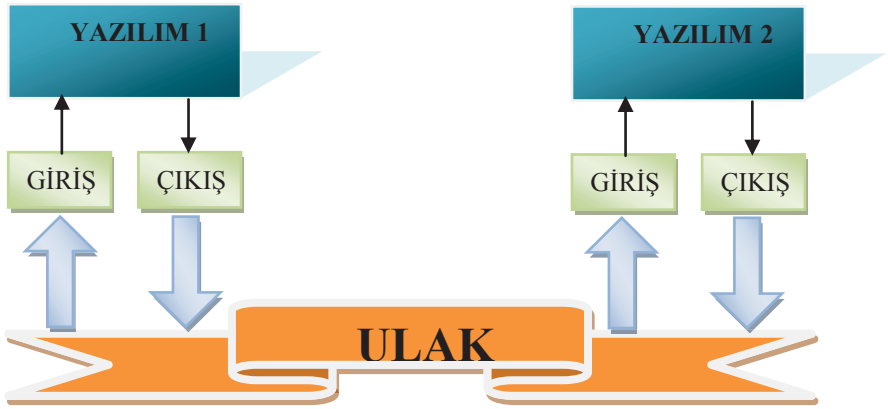
Şekil 1. Sistem yazılımı ULAK kullanımı ile üç seviyede yeteneklerini dış dünyaya açabilir

3.3 ULAK Desteğinin Otomatik Olarak Sağlanması

ULAK desteği yazılımlara otomatik olarak ULAK-Builder yazılımı ile eklenebilmektedir. Kullanıcının ULAK-Builder'a ULAK oluşturabilmesi için girdi olarak haberleşme arayüzlerini vermesi ve oluşturulacak UPA için isim belirlemesi yeterlidir.

3.4 İki Yazılım Arasında Haberleşme Ara Katman Rolü

ULAK, iki farklı yazılımın birbiri ile haberleşmesinin sağlanmasında kullanılabilir. Bu kullanım şekli ile ULAK, yazılımlar arasındaki haberleşmeyi otomatize ederken, Bölüm 3.1'de bahsedilen ara katman yeteneklerini de kazandırmış olur.



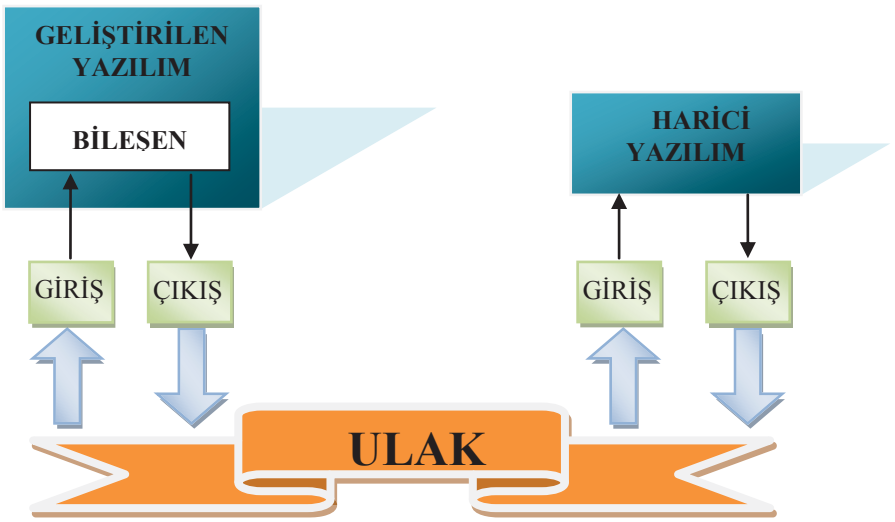
Şekil 2. İki yazılım arasında ULAK kullanımı

Şekil 2'de iki yazılımın ULAK kullanarak haberleşmesi görülmektedir. Yazılım 1, Yazılım 2'ye komut ya da veri göndermek istediğinde, çıkış üzerinden metot çağrısı yapar ve bu çağrı ULAK üzerinden geçirilerek Yazılım 2'ye iletilir.

3.5 Bileşenin Harici Yazılımlar Tarafından Kullanımı

Geliştirilen yazılımda kullanılan bileşenlere ULAK yeteneğinin kazandırılması durumunda, bu bileşen geliştirilen yazılımın içinden ya da harici bir yazılım tarafından Şekil 3'teki gibi kontrol edilebilir. Bunun için ULAK alt yapısı ile hazırlanmış bileşenlerden oluşan bir yazılımın kullanımı yeterli olacaktır. Yazılımın içindeki bileşenler, isteğe göre içeriden ya da dışarıdan yazılımların kontrolünde kullanılabilir. Bu tür kullanım şekline aşağıdaki kullanımlar örnek verilebilir:

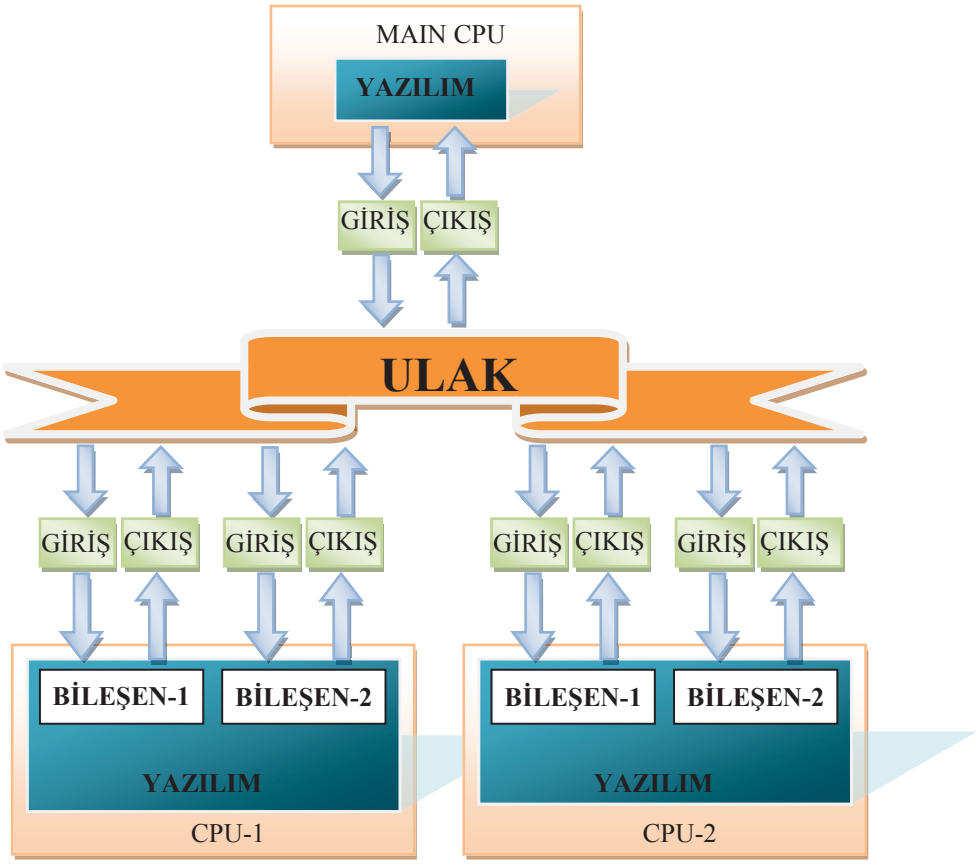
- Ana İşlemci Birimi Uygulama Yazılımı, başka bir CPU'da çalışan bir yazılımın haberleşme ve soyutlama bileşenlerine bağlanarak bu CPU'nun yeteneklerini kullanabilir.
- Donanım test çalışmaları kapsamında, harici bir test yazılımı, farklı bir CPU'da çalışan ULAK destekli bir yazılımın haberleşme ve soyutlama bileşenlerine bağlanarak bu CPU'nun yeteneklerini kullanabilir.
- İkmal bakım sistemleri için hazırlanmış olan Windows tabanlı test yazılımları, farklı bir CPU'da çalışan ULAK destekli bir yazılımın, haberleşme ve soyutlama bileşenlerine bağlanarak bu CPU'nun yeteneklerini, birim bileşenlerine bağlanarak ise tüm birimleri, arayüz fonksiyonları seviyesinde kullanabilir.



Şekil 3. Bileşenin harici yazılımlarla kullanımı

3.6 Yazılımın Bileşenler Bazında Farklı İşlemcilere Dağıtılması

ULAK altyapısı olan yazılımlar, birbirlerinin bileşenlerini kendi bileşenleri gibi kullanabilirler. Aşağıdaki şekilde görülen yazılımların tamamı, giriş/çıkış arayüz sınıflarının aynı olması şartıyla, aynı ya da farklı yazılımda olabilir. Şekil 4'te bu kullanıma örnek verilmiştir. Bu örnekte ana CPU'daki yazılım ULAK kullanarak iki farklı CPU'daki dört farklı bileşene görev dağılımı yapabilmektedir.



Şekil 4. Yazılımın bileşenler bazında farklı işlemcilere dağıtılması

Yazılımın, farklı işlemcilerdeki bileşenleri kullanabiliyor olması çeşitli faydalar ve kullanım alanları sağlamaktadır. Bu tür kullanım şekline aşağıdaki kullanımlar örnek verilebilir:

- Performans gereksinimi olan yazılımlarda, bileşenler farklı işlemcilere dağıtılarak paralel çalışmaları sağlanabilir.
- Yazılımın çalıştığı CPU'nun donanım kaynaklarının (port, NvRam, Flash vb.) yetersiz olması durumunda, başka CPU'ların kaynaklarını kullanabilmesine olanak sağlanabilir.

- Birbirine özdeş sistemler, birbirlerinin birimlerini (kamera, servo, gps, pusula vb.) sistemde fiziksel bir değişiklik yapmadan kullanabilirler.

3.7 Yazılımın / Bileşenin Kara Kutu Testi

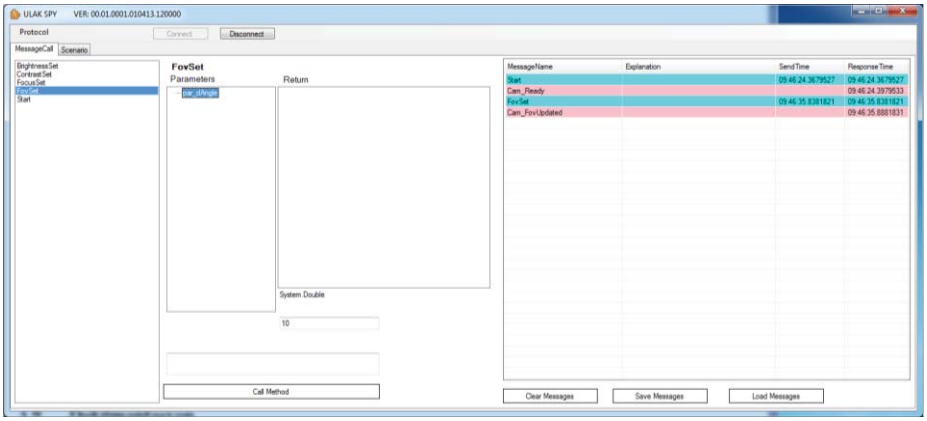
ULAK altyapısı olan yazılımların ve bu yazılımlarda bulunan yine ULAK altyapısı ile hazırlanmış tüm bileşenlerin kontrolünün harici yazılımların kontrolüne açılabiliyor olması, yazılıma yazılım ve bileşen seviyesinde kara kutu testi yapma imkanı sunmaktadır.

Windows tabanlı bir test yazılımı, ULAK arayüzünden yazılım seviyesinde ya da bileşen seviyesinde erişim sağlayarak, arayüz fonksiyonlarının izin verdiği detayda yazılımın verdiği tepkileri test edebilir. Bu yaklaşımla, yazılımların sadece yazılım seviyesinde değil, yazılımı oluşturan bileşen parçaları seviyesinde oldukça detaylı test edilmesi mümkün olur.

3.8 Yazılımın / Bileşenin Windows Kontrol Yazılımı

ULAK teknolojisi, arayüzlerin gerçeklenmesinin yanında, bu arayüzlere uygun Windows tabanlı kontrol yazılımlarının otomatik oluşturulmasını da sağlar. Bu sayede hem yazılım seviyesinde, hem de bileşen seviyesinde, tüm arayüzlere ait kontrol yazılımları otomatik çıkmış olur.

Şekil 5'te de görülebilen ULAK-SPY yazılımı, ULAK'ın hazırladığı arayüze ilişkin detayların yer aldığı ULAK-Builder tarafından otomatik oluşturulan DLL'i girdi olarak alır. Tüm arayüz yeteneklerini çalışma zamanında otomatik çözümler. İlgili arayüzü sağlayan kullanıcı arayüzünü kullanıcıya sunar.



Şekil 5. ULAK-SPY Yazılımı

3.9 Yazılımın / Bileşenin Windows Kontrol Yazılımıyla Testi

ULAK teknolojisi sayesinde, Bölüm 3.8'de tanıtılan ULAK-SPY ile test senaryoları yaratmak mümkündür. Bu arayüz ile test edilen yazılıma gönderilecek olan komutlar için çağrılmaya hazır fonksiyon setleri oluşturulabilir. Önceden hazırlanmış mesaj setleri istenilen sıra ile çağrılabilir. Bu sayede, geliştirme sürecinde, yazılımın geliştiriciler tarafından test edilmesi mümkün olur.

3.10 Dokümantasyon

ULAK teknolojisi, gerçeklediği arayüzlere ait mesaj ve yetenek listesini, mesaj formatını, mesaj tipini ve açıklamalarını otomatik olarak ayrıntılı şekilde oluşturur ve bir doküman ile kullanıcıya sunar. Bu doküman arayüz dokümanı olarak kullanılmaktadır. Örnek doküman Şekil 6'da görülebilir.

INPUT MESSAGES								
	Message Start :	Little Endian / Big Endian	Reserved	Call / Return	Message ID	Message Length	Data	Checksum
Value	:	Intel->L/Motorola->B	Reserved	C	ENUM_Camera_BrightnessSet		par_fBrightness	Sum of all bytes in the message
Size	1 Byte	1 Byte	1 Byte	1 Byte	4 Byte	4 Byte	4 Byte	4 Byte
ASCII Code		L: 0x4C B: 0x42		0x43	0x0(MsgID)	Length : 0x4	Type : float	Type : int
Description								
Value	:	Intel->L/Motorola->B	Reserved	C	ENUM_Camera_ContrastSet		par_fContrast	Sum of all bytes in the message
Size	1 Byte	1 Byte	1 Byte	1 Byte	4 Byte	4 Byte	4 Byte	4 Byte
ASCII Code		L: 0x4C B: 0x42		0x43	0x1(MsgID)	Length : 0x4	Type : float	Type : int
Description								
Value	:	Intel->L/Motorola->B	Reserved	C	ENUM_Camera_FocusSet		par_fFocus	Sum of all bytes in the message
Size	1 Byte	1 Byte	1 Byte	1 Byte	4 Byte	4 Byte	4 Byte	4 Byte
ASCII Code		L: 0x4C B: 0x42		0x43	0x2(MsgID)	Length : 0x4	Type : float	Type : int
Description								
Value	:	Intel->L/Motorola->B	Reserved	C	ENUM_Camera_FovSet		par_dAngle	Sum of all bytes in the message
Size	1 Byte	1 Byte	1 Byte	1 Byte	4 Byte	4 Byte	8 Byte	4 Byte
ASCII Code		L: 0x4C B: 0x42		0x43	0x3(MsgID)	Length : 0x8	Type : double	Type : int
Description								
Value	:	Intel->L/Motorola->B	Reserved	C	ENUM_Camera_Start			Sum of all bytes in the message
Size	1 Byte	1 Byte	1 Byte	1 Byte	4 Byte	4 Byte		4 Byte
ASCII Code		L: 0x4C B: 0x42		0x43	0x4(MsgID)	Length : 0x0		Type : int
Description								

Şekil 6. Otomatik olarak oluşturulan arayüz dokümanı

4 Sonuçlar

Dağıtık sistemlerde yazılımların birbirleriyle sağlıklı bir şekilde haberleşmesini sağlamak sistemler için hayati önem taşımaktadır. Mesaj çözümleme ve formatlama kodlarının elle yazılması iş gücü kaybına sebep olurken, sistemlerde haberleşme hatası oluşma riskini de arttırmaktadır.

Bu makalede özellikle kısıtlı kaynağa sahip, zaman-kritik dağıtık gömülü sistemlerde haberleşme için oluşturulmuş noktadan noktaya haberleşme teknolojisi olan ULAK ara katmanı aktarılmıştır. Farklı platformlar arasındaki haberleşmeyi sağlayabilmesi, uzak metot çağrısına imkan vermesi ve haberleşmeyi asenkron ya da senkron gerçekleştirebilmesi ULAK'ın başlıca özellikleri arasında sayılabilir.

5 Teşekkür

Gökhan Öztaş, Berkhan Deniz, Merve Özkardeş, Cansu Bender, Serdar Büyüksaraç, Bora Çalışkanbaş, Burak Ünaltay ve Volkan Şirin'e ULAK'ın geliştirilmesindeki destek ve katkıları için teşekkür ederiz.

Kaynaklar

1. S. Schneider and B. Farabaugh, "Is DDS for You?," A Whitepaper by Real-Time Innovations, http://www.rti.com/mk/DDS.html, 2009.
2. R. Kindel, "What Real-Time Data Distribution System Is Right for You?", AFRL Technology Horizons, August 2005
3. E. Curry, "Message-Oriented Middleware", Middleware for Communications, Q.H. Mahmoud, ed., John Wiley &, Sons, 2004
4. W. Emmerich, "Software engineering and middleware: a roadmap", Proceedings of the Conference on The Future of Software Engineering, p.117-129 , Limerick, Ireland, June 04-11, 2000
5. P. Boonma and J. Suzuki, "TinyDDS: An interoperable and configurable publish/subscribe middleware for wireless sensor Networks", in A. Hinze and A. Buchmann, editors, Principles and Applications of Distributed Event-Based Systems, chapter 9. IGI Global, 2010.
6. OMG, "Data distribution service for real-time systems – version 1.2", The Object Management Group, Tech. Rep., 2007.
7. P. Hintjens, "ZeroMQ: Messaging for Many Applications", O'Reilly, 2013.
8. P. Smolik, Z. Sebek, and Z. Hanzalek, "Orte–open source implementation of real-time publish-subscribe protocol", in Proc. 2nd International Workshop on Real-Time LANs in the Internet Age, pp. 68–72, 2003
9. Object Management Group, "The Common Object Request Broker: Architecture and Specification Revision 2.2", 492 Old Connecticut Path, Framingham, MA 01701, USA, February 1998.
10. L. Gilman and R. Schreiber, "Distributed Computing with IBM MQSeries", Wiley, 1996.
11. M. Hapner, R. Burridge, and R. Sharma, "Java Message Service Specification. Technical report, Sun Microsystems", http://java.sun.com/products/jms, Nov. 1999
12. S. Mishra, L. Fei, and G. Xing, "Design, implementation and performance evaluation of a corba group communication service", In Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, FTCS '99,, Washington, DC, USA, 1999. IEEE Computer Society.
13. D. C. Schmidt , A. S. Gokhale , T. H. Harrison , G. Parulkar, A high-performance end system architecture for real-time CORBA, IEEE Communications Magazine, v.35 n.2, p.72-77, February 1997

14. A. S. Gokhale and D.C. Schmidt, “Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks,” IEEE Transactions on Computers, vol. 47, pp. 391–413, 1998.