Dağıtık Sistemlerde Bileşen Tabanlı Bir Yazılım Mimari Çerçeve Önerisi

Emre H. KÖK[1] Özgür YILDIZ[2] Bekan ÇELİK[3]

[1,2,3]Barış Kartalı Projesi, HAVELSAN A.Ş., ANKARA

[1]e-posta: ekok@havelsan.com.tr [2]e-posta: oyildiz@havelsan.com.tr

[3]e-posta: bcelik@havelsan.com.tr

Özet

Dağıtık bilgi işleme mimarisi üzerine geliştirilen uygulamaların sayısı her geçen gün artmaktadır. Gerçek zamanlı CORBA ara katmanı üzerinde arayüz ve bileşen servisleri sağlayan çok katmanlı bir yapı, özellikle askeri projeler için çok önemli olan verimlilik, tutarlılık, ölçeklenirlik, ve güvenilirlik konuları göz önüne alındığında başarılı bir yazılım mimarisi tercihidir. Bu tür büyük ölçekli projelerde yazılım mimarisi için bir diğer önemli faktör de işletim sistemi bağımsızlığı ve geliştirilen yazılımların taşınılabilirliğidir. Bu bildiride, ACE üzerinde gerçek zamanlı CORBA servisleri kullanan ve genel arayüz ve bileşen servislerini gerçekleştiren bir üst katmanı ihtiva eden bir yazılım mimari çerçeve önerilmektedir. Bu yaklaşımla, bileşen tabanlı sistemlerin geliştirilmesi için bir altyapı oluşturulması ve geliştirilen bileşenlerde kod ve tasarım yeniden-kullanımı getirilerinden yararlanılabilmesi hedeflenmiştir.

Abstract

An increasing number of applications are being developed using distributed object computing middleware, such as CORBA. A real-time CORBA (RT-CORBA) based middleware supported by interface and component services layers provide enhanced efficiency, predictability, scalability, and reliability especially for military applications. Operating System abstraction and portability of the developed software is also another important factor in these large-scale applications. This paper presents a software architecture framework using RT-CORBA over ACE with interface and component services which consists of design patterns with benefits of code reuse, reduced system complexity and resources.

1. Giriş

Dağıtık programlama, dağıtık, açık, ölçeklenir, saydam ve hataları giderebilen bir programlama modelidir. Uzak yordam çağrıları (RPC), işletim sistemi komutlarını bir ağ bağlantısı üzerinde dağıtmak için kullanılır. CORBA, Microsoft D/COM, Java RMI gibi sistemlerin amacı, nesneye yönelik tasarımı, ağ üzerinde gerçekleştirmektir. Dağıtık bilgi işleme mimarisi, yazılım birimlerinin bir arada ama birden fazla bilgisayar sistemi üzerinde çalışacak şekilde tasarlanması fikrine dayanır. Bu mimari, çeşitlilik, ölçeklenirlik, yedeklilik, örgütleşim düzeni ve maliyet etkinliği gibi birçok

ihtiyacı karşılar. Böylece dağıtık sistem, farklı bilgisayarlar üzerinde koşut işletilen bileşenlerden oluşan uygulama olarak tanımlanabilir. Bu bileşenler, haberleşebilir ve ayrı ayrı işletilebilir olmalıdırlar.

Her proje, kendi içinde farklı sistem ve yazılım gereksinimleri içerse dahi, tüm projelerde belli başlı benzer altyapı gereksinimleri ile karşılaşılır. Bu ihtiyaçların başlıcaları; işletim sistemi taşınılabilirliği, haberleşme yönetimi, servis ilklenmesi, çoklanmış olay çözümü, olay işlenmesi, çoklu kullanım, eşzamanlılık, hata yönetimi, hata tolerans yönetimi ve bunlara ek olarak hız ve kaynak kullanımı gibi bazı verimlilik konularıdır. Özellikle dağıtık sistem tasarımında, bu ihtiyaçları karşılamak için ortak bir çözüm altyapısı oluşturma gerekliliği ön plana çıkmaktadır. Bu altyapıyı oluşturmak için geliştirilen bir mimari çerçeve üzerinde, benzer alanlara yönelik birçok uygulama ve proje geliştirilmesi, kod ve tasarım yeniden-kulanımını enbüyütebilecektir. Doğruluğu onaylanmış bir çerçeve sayesinde, geliştirilen uygulamaların güvenilirliği artacak ve test edilebilirlikleri kolaylaşacaktır. Askeri projelerin büyüklüğü ve ihtiyaçları göz önüne alındığında, yazılım tasarımının dağıtık bilgi işleme altyapısı üzerine kurulmuş bir yazılım mimari çerçeve ile gerçekleştirilmesi, zaman ve kaynak kullanımını çok daha etkin hale getirecektir.

1.1. Yazılım Mimari Çerçevesi (Software Architecture Framework, SAF)

Yazılım çerçevesi, birbirleriyle ilişkili uygulama aileleri için yeniden-kullanılabilir mimari üretmek amacıyla beraber çalışan tümleşik bileşenler kümesi olarak tanımlanabilir [1]. Çerçeve, genellikle çok sayıda tasarım örüntüsü içeren, büyük çaplı mimari bir yazılım öğesidir. Çerçeve, nesneye yönelik bir programlama diliyle gerçekleştirilmesi gerekmese de, tanımı gereği nesneye yöneliktir.

Sistem yazılımlarının bir çerçeve üzerine inşa edilmesinin getirdiği faydalar şu şekilde özetlenebilir:

- Tasarım ve kod yeniden-kullanımı sağlanır.
- Yazılımı geliştirenler, nesneler arası iletişim karmaşıklığıyla uğraşmaktan kurtulurlar ve genel olarak tüm sistem karmaşıklığı da azalır.
- Tüm geliştiriciler için ortak bir işlevsellik kümesi elde edilir.
- Geliştiriciler, iş mantığı tasarımı için daha çok zaman kazanırlar, sistem geliştirme süreci hız kazanır.
- Çerçeve, sistem kaynaklarının verimsiz kullanımını büyük ölçüde azaltarak sistem başarımını öngörülebilir kılar.

Sözü edilen tüm bu olanakları sağlamak amacıyla önerilen çözüm, SAF adı verilen, C++ programlama dili ve altyapısı üzerine kurulmuş, nesneye yönelik ve bileşen tabanlı bir mimari çerçevedir. SAF üzerinde geliştirilen bileşenler, proje ve sistem ihtiyaçlarına yönelik belirli bir alt sistemi veya bir parçasını gerçekleyen ve tanımlı arayüzleriyle birbirleriyle haberleşen yazılım birimleridir.

2. SAF

2.1. SAF Genel Yapısı

SAF, işletim sistemi ile uygulama katmanı arasında yer alan 4 ana katmandan oluşmaktadır.

1. İşletim Sistemi Soyutlama Katmanı: Geliştirilen yazılımların işletim sisteminden bağımsız bir şekilde kodlanabilmesi ve uygulamaların taşınılabilirliği, yazılımın tasarım aşamasında anahtar

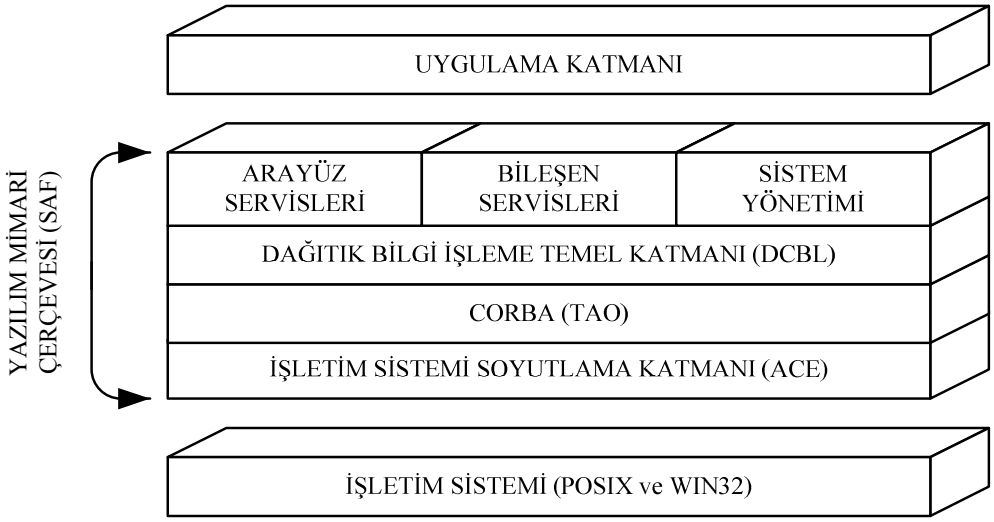
noktalardan birini oluşturmaktadır. C++ için standart kütüphane ve derleyiciler (ANSI C++) olmasına rağmen, işletim sistemi uygulama programlama arabirimlerinin (API) doğrudan kullanımını gerektiren kod parçalarında, işletim sistemi bağımlılığı kaçınılmazdır. Bu bağımlılığı, Prof. Douglas C. Schmidt ve Distributed Object Computing (DOC) grubu tarafından geliştirilen ADAPTIVE Communication Environment (ACE) [2] kullanımı ortadan kaldırmaktadır. ACE, işletim sistemi uygulama programlama arabirimleri üzerine yazılmış örtü sınıfları haricinde sağladığı üst seviye örüntü ve servislerle başlı başına bir çerçevedir. Ayrıca, birçok büyük dağıtık sistemin de temelini teşkil etmektedir.

2. CORBA: Temel bir tasarım kararı olarak, dağıtık bilgi işleme kullanımı yaygın olarak tercih edilen bir yöntemdir. Bu yöntem, dağıtık bileşenlerin farklı sorumlulukları gerçekleştirecek şekilde tasarlanıp geliştirilmesi, tümleştirilmesi ve farklı süreçler halinde birden fazla bilgisayarda çalıştırılabilmeleri mantığı üzerine kurulmuştur. Object Management Group (OMG) tarafından 1989'da geliştirilen Common Object Request Broker Architecture (CORBA) [3], dağıtık mimarilerde kullanılan en popüler ara katman yazılımlarından birisidir. CORBA servisleri ve altyapısı, farklı Object Request Broker (ORB) sağlayıcıları (TAO [4], ORBIX, OrbExpress, v.b.) tarafından gerçekleştirilmiştir.

3. Dağıtık Bilgi İşleme Temel Katmanı (Distributed Computing Base Layer, DCBL): Her ne kadar CORBA tek başına bir ara katman yazılımı olsa da, bileşen geliştiriciler için karmaşık CORBA servislerinin doğrudan kullanımını kolaylaştırmak, standartlaştırmak ve üst seviyede yazılan bileşen kodunun, değişik ORB tercihlerinden soyutlanmasını sağlamak amacıyla bu katmana ihtiyaç duyulmaktadır.

4. Genel Servisler ve Sistem Yönetimi Katmanı: Bileşen tabanlı mimaride, bileşenlerin ortak kullanabileceği arayüz, işlevsellik, ve sistem yönetimi servislerinin (yönetici ve aracı servisleri) sağlandığı katmandır.

SAF'ın genel yapısı Şekil 1'de gösterilmiştir.



Şekil 1. SAF genel yapısı

2.2. ACE

ACE, koşut iletişim için birçok temel tasarım örüntüsünü (olay işleme, çoklu kullanım, kilitleme, yapılandırma ve bağlantı gibi) gerçekleştiren ücretsiz, açık kaynak kodlu, taşınabilir, nesneye yönelik bir yazılım çerçevesidir. Yüksek başarımlı CORBA gerçekleştiriminin temelini oluşturmaktadır. ACE; süreçler arası iletişim, çoklanmış olay çözümü, belirtilmiş dinamik bağlantı ve koşutluk (concurrency) gibi kavramları temel alan nesneye yönelik ağ uygulamaları ve servislerinin geliştirimini, platform bağımlılığını ortadan kaldıracak şekilde başarır. Buna ek olarak, sistem yapılandırma ve yeniden-yapılandırmasını; servisleri, uygulamalara yürütüm esnasında dinamik olarak bağlayarak ve bu servisleri bir veya birden fazla süreç veya izlek içerisinde çalıştırarak otomatikleştirir [2].

2.3. CORBA

Dağıtık sistem tasarımı, ortak ağ programlama tekniklerinin güvenli olmaması, heterojen ağlar üzerindeki dağıtımın taşınabilirliği engellemesi, mevcut ağ ve soket programlama kütüphanelerindeki nesneye yönelik tasarım desteği eksikliği gibi nedenlerle karmaşık bir yapıdadır ve hatalara açıktır. Dolayısıyla, dağıtık sistemlerde karşılaşılan bu sorunların çözümü için nesneye yönelik bir mimari çerçeve tasarımına ihtiyaç duyulmaktadır.

OMG tarafından 1989'da geliştirilen ve bir standart haline gelen CORBA [3], dağıtık mimarilerde kullanılan en popüler ara katman yazılımlarından birisidir. CORBA, nesneye yönelik bir sunucu-istemci mimari altyapısı sayesinde dağıtık nesnelerin ağ üzerinde Interface Definiton Language (IDL) aracılığıyla tanımlanmış arayüz fonksiyonları ile iletişimini sağlar. CORBA, dağıtık uygulama tasarımı ve inşası için model oluşturan, arayüz ve gerçekleştirimi birbirinden ayırma, çok biçimlilik ve kapsülleme kavramlarını destekleyen nesneye yönelik bir çerçevedir. CORBA, birçok yazılım ve platform üzerinde gerçekleştirilmiştir ve kodlama arayüzü, tüm gerçekleştirimler arasında uyumludur.

CORBA, arayüz ve gerçekleştirimi birbirinden kesin olarak ayırır. Tüm arayüz ayrıntıları herkese açıktır ve IDL diliyle yazılır. Gerçekleştirim ayrıntıları istemcilerden gizlenmiştir. Tüm veri dönüşümleri, ara katman tarafından gerçekleştirilir. Sunucu ve istemciler arasında standart bir iletişim protokolü kullanılır. Tüm uyumlu CORBA gerçekleştirimleri, The General Inter-ORB Protocol (GIOP) adı verilen protokolü kullanırlar. Farklı sağlayıcılar tarafından geliştirilen ORB'lar birbirleri arasında, GIOP mesajlarını kullanarak haberleşirler. Ayrıca, tüm CORBA uyumlu ORB'lar, GIOP'yi TCP/IP ile eşleştiren ve Internet Inter-ORB Protocol (IIOP) adı verilen protokolü de desteklemelidirler [5].

2.3.1. TAO (The ACE ORB)

TAO [4], gerçek zamanlı, CORBA 2.2 uyumlu ve büyük ölçüde genişleyebilir bir mimariye sahip olan ACE kullanılarak geliştirilmiş, ikinci kuşak bir ORB'dur. ACE'i de geliştiren ekip tarafından tasarlanmıştır. TAO, gerçek zamanlı sistemlerin zorunlu gereksinimlerini karşılamak amacıyla tasarlanmış olmasının yanı sıra, genel amaçlı CORBA uygulamaları için de kullanıma hazır bir yapıdadır. Servis Kalitesi (Quality of Service, QoS) diye adlandırılan gerçek zaman gereksinimlerini karşılamak üzere tasarlanan TAO'nun, yüksek oranda tutarlı, genişleyebilir, yapılandırılabilir, taşınabilir, yüksek başarımlı ve açık standartlara uyumlu bir yapısı bulunur.

2.3.2. ACE ve TAO

TAO'nun birçok bileşeni ve üretilen koçan ve çerçeveler, ACE tarafından sağlanan örüntü ve bileşenleri temel alır. TAO, işletim sistemi işlevlerine erişmek için, taşınabilir olmayan sistem çağrılarını yürütmek yerine, ACE'in yüksek başarımlı işletim sistemi uyarlama bileşenini kullanır. ACE, pek çok işletim sistemini desteklediğinden, TAO'nun başka bir platforma taşınması da ACE ile aynı ölçüde kolaydır.

2.4. Dağıtık Bilgi İşleme Temel Katmanı (DCBL)

Bu katman, CORBA'nın karmaşık servis ve yapılarını soyutlayarak, geliştirilen bileşenler için sınıf hiyerarşisi ve tasarım altyapısı sağlar. Bileşen tabanlı bu mimaride üst seviyedeki bileşenlerin altyapısı (bileşenlerin kalıtım yoluyla türetilecekleri soyut sınıflar), bu katmanda gerçekleştirilmiştir. Bu katman, ACE ve TAO kütüphaneleri üzerine geliştirilmiştir, ancak altyapı itibarıyla bir başka ORB tercihinde de, üst seviyede yazılan bileşen kodunun kolayca uyarlanabilmesine imkan verir. Bu yapısı ile, TAO'dan farklı bir ORB tercihi söz konusu olduğunda, ORB'a özel kod parçalarının yalnızca bu katmana eklenmesiyle diğer bileşenleri bu farklılıktan soyutlamak hedeflenmiştir.

2.4.1. Bileşen Geliştirme Altyapısı

Bu mimaride bileşenler, dinamik olarak bağlanabilir kütüphaneler halinde geliştirilir ve her bir bileşen "*safserver*" isimli, çalıştırılabilir program sayesinde başlatılır. Bütün bileşenler en az bir adet lider CORBA nesnesi içerir. Lider bir nesne, kendisine ait nesne referansını, bir CORBA servisi olan isim sunucusuna, yalnızca kendisine ait bir isim ile kayıtlar. Bütün bileşenler, lider nesnelerin nesne referanslarına isim sunucusu aracılığıyla ulaşır. Bu işlevselliği gerçekleyen temel sınıf ve yöntemler DCBL katmanı içindedir. Üst seviyede yazılan bileşenlerde lider nesne arayüzü "*SAF::Component*" arayüzünden, nesnenin kendisi de "*SAF::Component_impl*" sınıfından türetildiğinde, otomatik olarak yukarıda bahsedilen yetenekler taşınmış olur. Lider nesneler, "*SAF::Component_impl*" içinde sanal olarak tanımlanmış bazı yöntemleri (*preActivate, postActivate, preDeactivate*) gerçekleştirir ve bileşene ait diğer CORBA veya C++ nesneleri bu yöntemler sayesinde yaratılır.

2.4.2. Varlık (Entity)

İsim sunucusuna kayıtlanmayan ve lider nesne olmayan CORBA nesneleri bu isimle anılır. "*SAF::Entity*" arayüzü bu tür nesneler için geliştirilmiş bir soyut sınıf olup, içerisinde CORBA arayüz yönetimini gerçekleyen işlevselliği barındırmaktadır. IDL ile tanımlanan arayüzler "*SAF::Entity*" arayüzünden ve bu arayüzleri gerçekleyen sınıflar da "*SAF::Entity_impl*" sınıfından türetildiğinde kalıtım yoluyla bu yetenekler taşınmış olur.

2.4.3. ORB Soyutlama (Bus)

ORB kavramını kapsülleyen yapıyı oluşturur. ORB'un sağladığı işlem ve servislerin yapılandırılması ve süreç ile ilişkisinin kurulması "*bus*" sayesinde gerçekleşir. Bileşenler içindeki CORBA nesnelerinin isim sunucusuna bağlanıp kayıtlanması için gerekli içeriği sağlar.

2.5. Genel Servisler ve Sistem Yönetimi Katmanı

Bu katman, uygulama bileşenlerinin ortak kullanabileceği arayüz, işlevsellik ve sistem yönetimi servislerinin gerçekleştirildiği birimlerden oluşmaktadır. Bu mimari çerçeve ile birlikte öngörülen çözüm, tüm bileşenlerin tasarımında ve kodlanmasında, geçerli bir ihtiyaç duyulmadıkça, bu katmanın sağladığı hizmetlerin kullanılmasıdır. Bununla beraber, SAF, daha alt katmanlarda bulunan kütüphane ve hizmetlerin doğrudan kullanımına da imkan vermektedir.

2.5.1 Arayüz Servisleri (Interface Services, INTSVC)

Bu serviste geliştirilmiş arayüz örüntüleri, CORBA olsun olmasın, tüm arayüzler için ortak bir davranış biçimi tarif eder. Bu birimde, bu örüntülere ait CORBA ve soket arayüzleri için gerçekleştirimler mevcuttur. Donanım arayüzleri bu birim kapsamında ele alınmamıştır. Her bir donanım arayüz tipini, CORBA arayüzüne dönüştüren uyarlayıcı bir servis bileşeni de ayrıca geliştirilmiştir.

Komut örüntüsü hem soket hem de CORBA arayüzleri için gerçekleştirilmiştir. Servis sağlayan bir nesneden komut gönderme yoluyla bilgi alınması veya belirli bir görevin yerine getirilmesinin istenmesi için kullanılan arayüz örüntüsüdür. Servis sağlayan nesnenin, gelen komutu gerçekleştirip bir sonuç dönmesi, istenilen komutu aldığına dair bir geri-bildirimde bulunması ve herhangi bir cevap veya geri-bildirim dönmemesi seçeneklerinin sunulduğu üç çeşit gerçekleştirimi mevcuttur.

Gözlemci örüntüsü, GOF Sağlayıcı/Tüketici (Supplier/Consumer) [6] örüntüsünün gerçekleştirimidir. Veri sağlayan CORBA nesnesi kendisine tüketici olarak kayıtlanmış nesnelere, sorumlu olduğu veriye dair içeriği ve her türlü değişikliği göndermekle yükümlüdür. Çoğa Gönderim (multicast) örüntüsü, gözlemci örüntüsünün, sağlayıcının veri dağıtımında kullandığı protokol altyapısı olarak çoğa gönderim tipini kullandığı biçimidir. Doğrudan erişimli gözlemci örüntüsünde ise gözlemci örüntüsünden farklı olarak bir veri grubuna ait tek bir sağlayıcı yerine, her bir veri nesnesinin kendi başına sağlayıcı olarak kullanılması durumu gerçekleştirilmiştir. Veri nesneleri kendilerine kayıtlanmış tüketicilere, içeriklerini ve güncelleme bilgilerini dağıtır.

2.5.2. Bileşen Servisleri (Component Services, CMPSVC)

Bileşen servisleri, geliştirilen bileşenler için uygulamaya özel olmayan genel ihtiyaçlara yönelik gerçekleştirimlerin yapıldığı bölümdür. Bu bölüm, alt katmanlarda (ACE, DCBL) yazılmış bazı servisler için yapılmış birtakım değişiklik ve düzenlenmeleri içerir. Genel yapı itibariyle STL sınıfları üstüne yazılmış örtü sınıflardan oluşur.

Bileşen servisleri içinde bulunan işlevler aşağıdaki şekilde özetlenebilir.

- Kap (Container): İzlek güvenliği eklenmiş STL kapları.
- Aykırı durum (Exception): C++ ve CORBA aykırı durumları için standart bir yapı oluşturulmuştur.
- Teklik (Singleton): GOF Teklik (Singleton) [6] örüntüsünün gerçekleştirimidir.
- Zamanlayıcı (Timer): Geri çağırım (callback) temelli bir zamanlayıcı gerçekleştirimidir. Belirli bir zamana programlanmış veya belirli sıklıkta çalıştırılacak yöntemlere olanak sağlar.
- İşler (Jobs): Mimari çerçeve içinde geliştirme modeli, genel olarak olaya dayalı iş kullanımı temeline oturtulmuştur. Olaya dayalı iş, bir olay meydana geldiğinde tanımlanmış bir görevin

uygulanmasıdır. Bu mimaride kullanılan iş çeşitleri; arayüzler üzerinden CORBA işlev çağırımları, işçiler, periyodik işçiler, tek kullanımlık işçiler olarak sıralanabilir.

- İzlek Altyapısı (Thread Infrastructure): ACE içinde gerçekleştirilmiş izlek altyapısına yapılan çeşitli düzenlemelerden meydana gelir. İşlerin ve zamanlayıcıların çalıştırılması için gerekli mekanizmalara ve tanımlanmış işler için belirli sayıda izlek içeren havuz yapılarına olanak sağlar.
- Eşzamanlılık (Synchronization) Örüntüleri: Çoklu izlekli uygulamalarda verinin tutarlılığını koruyan kilitleme mekanizmalarını gerçekleştirir. Mutex, RWMutex, Latch, Semaphore, Barrier gibi sınıflar sağlanmaktadır.
- Akıllı İşaretçiler (Smart Pointers): Akıllı işaretçiler, bellek yönetimi için kullanılan yaygın bir C++ tekniğidir. Aynı şekilde, referans sayma yöntemi de nesnelere ait yaşam döngülerini yönetme, bellekten yer alma ve geri bırakma işlemleri için güçlü bir mekanizmadır. Anlamsal (semantic) olarak CORBA *var* örüntüsüne benzeyen bir sınıf çerçevesi, CMPSVC tarafından sağlanmaktadır.
- Dosya İşlemleri (File I/O): "*libc*" dosya işlemleri uygulama programlama arayüzü üzerinde nesneye yönelik bir örtü teşkil eden sınıf yapılarından oluşmaktadır.
- Sistem Tip ve Araçları: Çeşitli matematiksel ve mantıksal hesaplamalara yönelik servisleri sağlayan birimdir. Dizey işlemleri, zaman ve tarih işlemleri, birim ve koordinat dönüşümü işlemleri gibi genel ihtiyaçlara yönelik servisler, bu birim tarafından gerçekleştirilmiştir.

2.5.3 Sistem Yönetimi

Bileşen tabanlı mimaride bütün bileşenleri çalıştıran ve bunların başarım, bellek yönetimi ve CPU kullanımını izleyip, gereken kayıt tutma ve kontrol mekanizmasını gerçekleştiren birimdir. Ayrıca, sistemin genel ihtiyaçlarına yönelik geliştirilmiş bileşenler, bu birim kapsamında ele alınmaktadır. Uygulamalar için geliştirilen bileşenler içinde kalıtım yoluyla türetilmiş aracı nesneler, sistem yönetimi bileşenleri içinde geliştirilmiş yönetici nesnelere kayıtlanıp, ilgili işlevlerin gerçekleştirilmesini sağlar.

Sistem yönetimi için geliştirilmiş servisler aşagıdaki gibi sıralanabilir.

- Sistem Kontrol: Bileşenlerin sistem genelinde davranışlarını kontrol eden, ilkleyen, sonlandıran, donanım ve yazılım aksaklıklarında bileşenleri kapatıp, yeniden başlatabilen mekanizmayı sağlar. SAF sunucusunun (*safserver*) gerçekleştiriminin yapıldığı birimdir.
- Sağlık Yönetimi: Başarım takibi, oluşan hata durumlarının algılanması ve düzeltilmesi gibi işlevleri içerir.
- Kontrol Noktası ve Normale Dönme: Bu sistem bileşenlerin içsel durumlarını ve içerdiği veriyi belirli zamanlarda kaydedip, istenmeyen sonlanma durumlarında, en az veri kaybı ile normale dönme özelliğini sağlar.
- Kaydetme ve Yeniden Oynatma: Uygulama anında yaşanmış bütün olayların, kullanıcı hareketlerinin, alınan ve üretilen verilerin yürütüm esnasında kaydedilip sonradan yeniden oynatılmasına imkan veren servisten sorumlu birimdir. Yeniden oynatma işlemine, operasyon sonrası analiz, başarım değerlendirme ve eğitim amaçları için ihtiyaç duyulmuştur.
- Zaman Yönetimi: Görev-sistem zamanı yöneticisi, sistem bütününde zaman bilgisi dağıtımından sorumlu birimdir. Bileşenler içinde sistem zamanı bilgisi bu servis içinde gerçekleştirilmiş tekil zaman aracı nesnesinden elde edilir. Bu tekillik yapısı, sistem genelinde bir zaman tutarlılığı sağlar. Örneğin, yeniden oynatım anında iki kat hızlı bir izleme ihtiyacını karşılamak için, zaman yönetimi servisinin, bir birim zamanı yarım birim zaman olarak dağıtacak şekilde yapılandırılması yeterli olacak, belirli zaman aralıklarında çalışan izleklerin

otomatik olarak çalışma aralıkları azalacak ve iki kat hızlı oynatma işlemi, ek bir araç ve gerçekleştirmeye gerek duyulmadan sağlanmış olacaktır.

- Yük Yönetimi: Yük yönetiminde amaçlanan, özellikle sistemdeki kritik işlemler için işlemsel olarak kabul edilebilir servis ve hız kalitesinin sağlandığının takip ve denetiminin yapılmasıdır.
- Alarm Yönetimi: Özel durumlarda kullanıcıya çıkması gereken alarm ve uyarıların ortak bir birim aracılığıyla işlenmesi için geliştirilmiş bir servistir. Alarmların kullanıcıya gösterilmesi, önceliklerinin belirlenmesi, alarmların silinmesi ve kaydedilmesi işlemlerini içerir.

3. Sonuç

C^4ISR (Command, Control, Communications, Computer, Intelligence, Surveillance and Reconnaissance - Komuta, Kontrol, İletişim, Bilgisayar, İstihbarat, Gözetleme ve Keşif), askeri ihtiyaçları karşılayan araçlar, bilgisayar ve iletişim donanımları, görüntü aygıtları ve istihbarat sistemlerinin tümü olarak tanımlanmaktadır. Bu tarz büyük sistemlerde, sistemi oluşturan her bir donanım ve yazılım biriminin, genellikle belirli bir işlevi gerçekleştirmek üzere değişik bölgelerde konuşlandırılma ve tümleşik olarak çalıştırılma ihtiyacı bulunmaktadır. Bu gereksinimlerin karşılanması için, dağıtık sistem altyapısının tercih edilmesi zorunluluğu ortaya çıkmıştır. Tasarlanacak sistem ilk olarak, sistem mühendisliği tarafından alt sistem ve bu alt sistemi oluşturan donanım ve yazılım bileşenlerine ayrılır. Alt sistemi gerçekleyen bir yazılım birimi, bilgisayar Yazılım Konfigürasyon Parçası (YKP, CSCI), YKP'yi oluşturan her bir yazılım birimi de bilgisayar Yazılım Konfigürasyon Bileşeni (YKB, CSC) olarak adlandırılır. YKB'ler, dağıtık bilgi işlemenin gerektirdiği, alt sistemin belli bir işlevini gerçekleştiren, tanımlı bir arayüzü bulunan, ayrı olarak çalıştırılabilen, alt sistemin diğer parçalarıyla tümleşik bileşen tanımına uygundur. Yazılım tasarım süreci bu bileşenler arasındaki arayüzlerin belirlenmesiyle başlar, bileşenlere ait tanımlı görevlerin gerçekleştirimine yönelik içsel tasarımlarıyla devam eder. Gerek arayüz tasarımları, gerekse içsel tasarımlar, bir mimari çerçeve temeli üzerinde gerçekleştirilir. SAF, çerçeve kullanımının getirdiği faydaların ve bir sistem tasarımı için altyapı oluşturmasının yanı sıra, yazılım geliştirme takımlarının ihtiyaç duyduğu bir yazılım geliştirme sürecinin tanımlanmasına da katkıda bulunmaktadır.

Barış Kartalı Projesi, Türk Hava Kuvvetleri'nin kullanacağı Havadan Erken İhbar ve Kontrol (Airborne Early Warning and Control, AEW&C) sisteminin geliştirilmesi projesidir. HAVELSAN A.Ş., Türk AWACS uçağı (Airborne Warning and Control System, Havadan Erken İhbar Komuta Kontrol Sistemi) da denilen "Barış Kartalı" uçağının görev bilgisayarı yazılımlarını geliştirme görevini üstlenmiştir. Yazılımlar, bu bildiride sunulan yazılım mimari çerçevesine benzer bir yapı üzerinde ve yine bu çerçeve üzerinde tarif edilmiş bir yazılım geliştirme sürecine uygun şekilde geliştirilmektedir.

Türk Silahlı Kuvvetleri'nin ihtiyaç duyduğu C^4ISR bilgi sistemleri geliştiriminde, verimli, yüksek oranda tutarlı, ölçeklenir, güvenilir, yapılandırılabilir ve taşınabilir yazılım çözümlerine gidilmesi gerekliliği kaçınılmazdır. Dağıtık bilgi işleme mimarisini temel alan bir yazılım mimari çerçeve altyapısı üzerine oturtulmuş standart bir yazılım süreci, birbirine benzer projeler için ortak ve maliyet etkin bir çözüm sunabilecektir. Böyle bir çözümün kurumsallaşmasının; şirket içi birlikteliğe (sinerji), müşteri ihtiyaçlarının karşılanmasına, kalite hedeflerine ulaşılmasına ve şirket misyonunun yerine getirilmesine önemli katkıları olacaktır.

Kaynakça

[1]. Houston S. D., Johnson J. CE ve Syyid U., The ACE Programmer's Guide, Pearson Education Inc., A.B.D., 2003.

[2]. Douglas C. Schmidt'in ACE internet sitesi: http://www.cs.wustl.edu/~schmidt/ACE.html

[3]. Douglas C. Schmidt'in TAO internet sitesi: http://www.cs.wustl.edu/~schmidt/TAO.html

[4]. OMG'nin CORBA internet sitesi: http://www.corba.org

[5]. Henning M. ve Vinoski S., Advanced CORBA Programming with C++, Addison-Wesley, A.B.D., 1999.

[6]. Gamma E., Helm R., Johnson R. ve Vlissides J., Design Patterns., Addison-Wesley, A.B.D., 1995.