2006

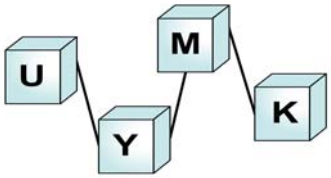
Ulusal Yazılım Mimarisi Konferansı

20-21 Kasım 2006
İstanbul, Türkiye

Bildiriler Kitabı

Editörler

Bedir Tekinerdoğan, Twente Üniversitesi
Oya Kalıpsız, Yıldız Teknik Üniversitesi
Semih Çetin, Cybersoft
Ali Doğru, Orta Doğu Teknik Üniversitesi



2006

Ulusal Yazılım Mimarisi Konferansı

20-21 Kasım 2006
İstanbul, Türkiye

Bildiriler Kitabı

Editörler

Bedir Tekinerdoğan, Twente Üniversitesi
Oya Kalıpsız, Yıldız Teknik Üniversitesi
Semih Çetin, Cybersoft
Ali Doğru, Orta Doğu Teknik Üniversitesi

DESTEKLEYİCİ KURULUŞLAR

Microsoft®

Cybersoft
vision@work to make IT happen

IEEE
TURKEY SECTION

NBA
NETRON
BİLİŞİM
AKADEMİSİ

Bildem
@LIFECYCLE ENGINEERING COMPANY

İÇİNDEKİLER

Davetli Konuşmacılar

Bildiri Oturumları

Uygulama Mimarisi ve Altyapı – 1

Bileşen Yönelimli Yazılım Tasarımı

Uygulama Mimarisi ve Altyapı – 2

Modelleme

Referans Mimarisi ve Uyarlanabilirlik

Tasarım Desenleri

Yazılım Mimarisinde Kalite

Model-Güdümlü Mimari

Endüstri Oturumu

Posterler

ÖNSÖZ

Geçen on yılda yazılım mimarisi kavramı oldukça güncellik kazandı ve geniş ölçekli karmaşık yazılım sistemlerinin geliştirilmesindeki zorluklarla başa çıkmak için ana rol oynayacağı genel bir kanı haline geldi. Yazılım mimarileri erken tasarım kararlarını içerir ve bütün sistemin kalitesini etkileyecek genel yapıyı ortaya koyar. Uygun bir mimari olmadan yüksek kalitede bir yazılım sistemi elde etmek zordur.

Geçen yıl Ulusal Yazılım Mühendisliği Sempozyumu (UYMS 2005) kapsamında Yazılım Mimarisi Çalıştayı'nı organize ettikten sonra bu yıl Yazılım Mimarisi konusu üzerine ilk kez ulusal bir konferans düzenlemeye karar verdik. Ulusal Yazılım Mimarisi Konferansı (UYMK) yalnızca yazılım mimarisi ve ilgili konuları içermektedir ve Türkiye'de yazılım mühendisliği konusu ile ilgili pratisyenleri ve akademisyenleri, her yönüyle yazılım mimarisi ile ilgili deneyim, sonuç ve fikir paylaşımı yapmak üzere bir araya getirmeyi amaçlamaktadır. Yazılım mimarisi tasarımı ile ilgili bu konferansın temel hedefleri aşağıda sıralanmıştır:

- *Yazılım mimarisi tasarımı bilincini geliştirmek*

Temel hedef, Yazılım Mühendisliği için bir anahtar kavram olan bu alanı geliştirmektir. Bu hedefi desteklemek üzere konferansın bildiri oturumlarında yazılım mimarisi tasarımının değişik konuları ele alınmaktadır.

- *Yazılım mimarisi tasarımı konusunda araştırma ve eğitimi özendirmek*

Yazılım mimarisi tasarımı araştırma ve eğitiminin hareketlenmesi umulmaktadır. Bi konferansta araştırmacılar, fikirlerini sunma ve paylaşma ortamı bulacaklardır. Eğitimciler ise derslere eklenmek üzere yazılım mimarisi tasarımında önemli olan konuları belirleyebileceklerdir.

- *Yazılım mimarisi tasarımının uygulanmasını özendirmek ve tartışmak*

Yazılım mühendisliği ve yazılım geliştirme, Türk toplumunda bir anahtar rol üstlenmek üzeredir. Geçen on yıl zarfında ciddi uluslararası rekabet kabiliyeti olan birçok saygın yazılım kuruluşu ortaya çıktı. Bu şirketlerin genelde uluslararası yazılım mühendisliği camiası ile bir uyum sağladığı gözlemlenmektedir. Bu konferans ile yazılım mimarisi tasarımı konusunda bir durum değerlendirmesinin oluşturulabileceğini umuyoruz. Konferans, endüstrideki en son gelişmeleri temsil ederek belirlenmiş problemler ve çözümlerini de öne çıkarmak için bir fırsat oluşturmaktadır.

Konferansın Program Komitesinde hem akademik çevrelerin hem de yazılım sektörü temsilcilerinin yer almasına önem verilmiştir. Konferansa gönderilen 36 bildiri arasından 24 bildiri, Program Komitesinin değerlendirmeleri sonucunda konferansta sunulmak üzere kabul edilmiştir. 6 bildiri ise poster gösterimi için kabul edilmiştir. Kabul edilen bildiriler yazılım mimarisi konusunda, Model-Güdümlü Mimari, Servis-Odaklı Mimari, Yazılım Mimarisinde Kalite, Uygulama Mimarileri, Yazılım Mimarileri ve Tasarım Desenleri, Referans Mimarisi ve Uyarlanabilirlik, Yazılım Mimarisi ve Bileşenler, gibi farklı alanları içermektedir. Bildiri oturumların yanı sıra bir endüstri oturumu ve panel de organize edilmiştir. Endüstri oturumunda yazılım mühendisliğine ülke çapında önemli katkıda bulunan *ASELSAN*, *Cybersoft* ve *Logo Business Solutions* temsilcileri yazılım mimarisi alanındaki kendi uygulamalarını ve tecrübelerini anlatmışlardır.

Konferansta iki davetli konuşmacının sunum ve görüşlerine yer verilmiştir. Sayın Prof. Dr. Mehmet Akşit soyut ve belirsiz ortamlarda yazılım mimarisi kalitesinin tanımı ve iyilenmesi

hakkında bir sunum vermiştir. Sayın Dr. Naci Akkök ise Servis Odaklı Mimarinin önemi ve kullanım yöntemleri hakkında bilgi vermiştir.

Yazılım mimarisinin uluslararası platformda yıllardır giderek önem kazandığını düşünerek bu konferansın da ülkemizde yazılım mühendisliği disiplinin gelişmesine katkıda bulunulacağını ümit ediyoruz. Konferansın organizasyonunda emeği geçen tüm program komitesi üyelerine, davetli konuşmacılarımıza, Yıldız Teknik Üniversitesi yerel organizasyon kadrosuna, destekleyici kuruluşlarımıza ve konferansın tüm katılımcılarına teşekkürlerimizi sunarız.

Bedir Tekinerdoğan, Twente Üniversitesi
Oya Kalıpsız, Yıldız Teknik Üniversitesi
Semih Çetin, Cybersoft
Ali Doğru, Orta Doğu Teknik Üniversitesi

TEKNİK ORGANİZASYON KOMİTESİ

Bedir Tekinerdoğan, University of Twente, Hollanda (Genel Başkan)
Oya Kalıpsız, Yıldız Teknik Üniversitesi (Düzenleme Kurulu Başkanı)
Semih Çetin, Cybersoft
Ali Doğru, Orta Dogu Teknik Üniversitesi

YEREL ORGANİZASYON KOMİTESİ

Yıldız Teknik Üniversitesi

Songül Albayrak
Banu Diri
Özgür Bozkurt
Ayşe Buharalı
Zeyneb Kurt
Ekinsu Uğurlu
Hakan Haberdar

PROGRAM KOMİTESİ

Naci Akkök, University of Oslo, Norveç
Mehmet Akşit, University of Twente, Hollanda
Levent Alkışlar, ASELSAN
Ali Doğru, Orta Doğu Teknik Üniversitesi
Selim Akyokuş, Doğuş Üniversitesi
Meriç Aykol, TBD İstanbul
Turgay Aytaç, Logo Business Solutions
Fevzi Belli, Universität Paderborn, Almanya
Semih Çetin, Cybersoft
Oğuz Dikenelli, Ege Üniversitesi
Banu Diri, Yıldız Teknik Üniversitesi
Nadia Erdoğan, İstanbul Teknik Üniversitesi
Yenal Göğebakan, Cybersoft
Haluk Gümüşkaya, Fatih Üniversitesi
Oya Kalıpsız, Yıldız Teknik Üniversitesi
Alpay Karagöz, Bilgi Grubu
M. Ümit Karakaş, Başkent Üniversitesi
Sefer Kurnaz, Hava Harp Okulu
Can Özturan, Boğaziçi Üniversitesi
Özgür Ulusoy, Bilkent Üniversitesi
Coşkun Sönmez, Yıldız Teknik Üniversitesi
Murat Tanık, University Alabama at Birmingham, USA
Bedir Tekinerdoğan, University of Twente, Hollanda
Yasemin Topaloğlu, Ege Üniversitesi
Tunç Torosdağlı, Milsoft

Davetli Konuşmacılar

Soyut ve Belirsiz Ortamlarda Yazılım Mimarisi Kalitesinin Tanımı ve İyilenmesi

Prof. Dr. Mehmet Akşit

Chair Software Engineering
Computer Science, University of Twente
Enschede, The Netherlands
http://wwwhome.cs.utwente.nl/~aksit

Özet

Yazılım mimarisinin temel amacı gerekli kaliteleri sağlayan yazılım şeklinin ortaya çıkarılıp tanımlanmasıdır. Yazılım mimarisi tasarlayanların bu amacı yerine getirmede karsılaştıkları önemli güçlükler vardır. Her şeyden önce arzu edilen kalite değerinin somut olarak tanımlanması gerekir. Kalite değerleri somut olarak modellense bile, soyut olan ön tasarım süreçlerinde daha ileride gerçekleştirilecek olan yazılımın kalite değerlerinin ölçülmesi zordur. Üçüncüsü, birbirine çelişkili olan kalite değerleri arasındaki dengeyi sağlamak için elimizde yeterli teknikler bulunmamaktadır. Üstelik gerek gereksinmelerin tanımlanmasında gerekse tasarım surecinde doğal olarak mevcut olan belisizlikler tasarımın başarısını olumsuz yönde etkilemektedir.

Bu koşullarda yazılım mimarisi kalitesinin tanımı ve en iyilenmesinin sağlanması için yeni yöntem ve tekniklerin geliştirilmesi gerekmededir. Sunumda önce mevcut kalite modelleme ve dengeleme yöntemleri tanıtılacak ve eksiklikleri vurgulanacaktır. Daha sonra Twente Üniversitesi yazılım mühendisliği bölümünde geliştirilen soyut ve belirsiz ortamlarda yazılım mimarisi kalitesinin en iyilenmesi teknikleri pratik örneklerle açıklanacaktır.

A Practical and Methodical Interpretation of the Service Oriented Architecture

Dr. M. Naci Akkök

Chief Architect, Oracle Nordics, and Assoc. Prof. II,
Informatics Department,
University of Oslo, Norway

Abstract

As with any architectural style, the Service Oriented Architecture (SOA) implies a design & development paradigm that manifests itself as a method framework. The SOA method framework, in turn, implies what tools are needed in the SOA stack to support the method fully. In this session, I will present a unique methodical (business process & model centric) interpretation of SOA as well as the "ideal" SOA tool stack to support the SOA method framework implied by the interpretation. I will also show briefly how far the "ideal" SOA tool stack is implemented today by mapping the ideal onto one major vendor's actual SOA product stack (i.e., Oracle® Fusion Middleware).

Uygulama Mimarisi ve Altyapı - I

Komuta-Kontrol Sistemleri için Ulusal Yazılım Mimarisi Geliştirme Projesi

Ediz ŞAYKOL, Devrim SARAN, İsmail KILINÇ,
Mehmet YAZGEÇ, Menderes AYDIN, Özgür YILDIZ
HAVELSAN A.Ş. Eskişehir Yolu 7. Km 06520 ANKARA
{esaykol, dsaran, ikilinc, myazgec, maydin, oyildiz}@havelsan.com.tr

1. Giriş

1.1 Temel Amaç

HAVELSAN A.Ş. bünyesinde geliştirilmeye başlanan, komuta-kontrol sistemleri için ulusal yazılım mimarisi geliştirme projesinin temel amacı, bu alandaki uygulamaların gereksinim duyacağı servisleri önceden belirli kalitede (Quality-of-Service) sunabilecek, gerçek zamana yakın, yüksek uygulama geliştirme üretkenliği sağlayan bir arakatman (middleware) geliştirilmesidir.

1.2 Teknik Bilgiler

ObjectWeb konsorsiyumu [1], aratkatman tanımını şu şekilde vermektedir: "*Bir dağıtık (distributed) bilgi işleme sisteminde, işletim sistemiyle, sistemin her bir bilgi işlem noktasındaki (site) uygulamalar arasında yer alan yazılım katmanına arakatman adı verilir*". Nesne tabanlı ve olay (event) tabanlı olmak üzere temel olarak iki adet arakatmandan söz edilebilir. Nesne tabanlı arakatmanlar, istek/yanıt (request/reply) biçemli, uzak yordam çağrısı (remote procedure call, RPC) tabanlı ve eşzamanlıdırlar (synchronous). Bu tür arakatman yazılımlarına CORBA [2] ve D/COM [3] örnek olark verilebilir. Olay tabanlı arakatmanlar, tek kerelik (single shot) iletişim için global ve gezici (mobile) sistemler, dağıtık aktif sistemler (yayın/abone) ve mesaj yönelimli sistemlerde kullanılan arakatmanlardır. DDS [5], örnek olarak verilebilir.

Arakatmanların getirdiği faydalar, şu şekilde özetlenebilir:

- Dağılım gizlenmiştir. Bir uygulama, dağıtık lokasyonlarda koşan birçok birbirine bağlanmış parçadan oluşur.
- Çeşitli donanım bileşenleri, işletim sistemleri ve iletişim protokolleri arasındaki heterojenlik gizlenmiştir.
- Uygulama geliştiren veya entegre edenler için tekil, standart ve yüksek seviye arayüzler sağlanmıştır. Böylece, uygulamalar kolaylıkla oluşturulur, tekrar kullanılır, aktarılır ve bir arada çalışmaları sağlanır.
- Benzer çabalardan sakınılması ve uygulamalar arası işbirliğinin kolaylaştırılması amacıyla, birçok genel amaçlı fonksiyon için ortak bir servis kümesi sağlanmıştır.

Çerçeve, çeşitli arakatman mimari ve gerçekleştirimleri üzerine inşa edilmiş, özel bir tanım alanı için ihtiyaç duyulan tüm servisleri, bu mimari üzerinde gerçeklemek üzere tasarlanmış, birbirleriyle işbirliği yapan/etkileşen bir sınıflar topluluğudur. Özel uygulamalar üretmek için uyarlanmış, yeniden kullanılır, yarı-tamamlanmış bir altyapıdır ve tasarımı, soyut sınıflar halinde bölümleyen, bunların sorumluluk ve etkileşimlerini tanımlayan bir mimari tanımlamasıdır.

Çerçevelerin getirdiği faydalar, şu şekilde özetlenebilir:

- Tasarımların ve kodlamanın yeniden kullanımı sağlanır.
- Düşük sistem karmaşıklığı ve kaynak tüketimi sağlanır.
- Sistem geliştiricileri, sınıflar arası iletişim karmaşıklığından uzak tutulur. Geliştiriciler tarafından yazılması gereken kod, tahmin edilebilir ve standart olur.
- Tüm geliştiriciler için ortak bir işlevsellik kümesi sağlanır.
- Sistemin iş mantığını geliştirmek için daha fazla zaman ayrılacağı için, sistem geliştirimi hız kazanır.
- Sistem kaynaklarının olumsuz kullanımı önlenir.

Dağıtık programlama, dağıtık, açık, ölçeklenir, saydam ve hataları giderebilen bir programlama modelidir. Uzak yordam çağrıları, işletim sistemi komutlarını bir ağ bağlantısı üzerinde dağıtmak için kullanılır. CORBA [2], Microsoft D/COM [3], Java RMI [4] 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 şunlardır:

- işletim sistemi taşınabilirliği (platform independence/portability),
- gerçek/yarı-gerçek zamanlı işletim (real-time/near real-time processing),
- güvenilirlik/doğruluğu onaylanmış (reliability),
- genişletilebilirlik (extensibility),
- kullanılabilirlik (availability),
- sürdürülebilirlik (maintainability),
- değişik servis kaliteleri (various quality-of-service, Qos)
- bağlantı yönetimi (connection management),
- servis ilklenmesi (service initialization),
- çoklanmış olay çözümü (event demultiplexing),
- olay işlenmesi (event handler dispatching),
- çoklu kullanım (multi-threading),
- eşzamanlılık (synchronization),
- hata ve hata tolerans yönetimi (fault detection and fault tolerance),
- hız, performans (speed and performance),
- verimli kaynak kullanımı (resource management),
- kod ve tasarım yeniden kulanımı (code and design reuse),
- test edilebilirlik (testability).

Özellikle dağıtık sistem tasarımında, yukarıdaki 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 maksimize edebilecektir. Doğruluğu onaylanmış bir çerçeve sayesinde, geliştirilen uygulamaların güvenilirliği artacak ve test edilebilirlikleri kolaylaşacaktır. Askeri projelerin büyüklüğü ve gereksinimleri 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.

1.3. Tanım Alanı Veri Analizinin Önemi

Verilen bir tanım alanı içerisinde, verinin altsistemler ve dağıtık konuşlandırılmış yazılım bileşenleri arasında belirli servis kalitelerinde aktarılması büyük önem arz etmektedir. Gerçek ve/veya yarı gerçek zamanlı gereksinimlerin karşılanması için, donanım ve yazılım mimarilerinin, verilmiş olan tanım alanı için uygun olması gerekmektedir. Bu noktada, komuta kontrol tanım alanında kullanılacak bir çerçeve için bir ödünleşim (trade-off) ortaya çıkmaktadır. Birinci seçenek, tanım alanı veri analizi yapılmış bir sistem için arakatman tasarımı ve geliştirimidir. Bu durumda, seçilen yazılım mimarisinin ne olacağı kısmen belli olacağı için, o tanım alanına özel bir arakatman tasarımına gidilebilir. Ancak, gerçekleştirilen çerçevenin, diğer sistemlerde yeniden kullanımı söz konusu olmayabilir. İkinci seçenek ise, geliştirilecek çerçevenin, dağıtık mimarilerde yaygın olarak kullanılan arakatmanları (CORBA, DDS, v.b.) sarmalaması (encapsulation) ve soyutlaması (abstraction) yoluyla elde edilecek genel bir mimari tercih edilmesidir. Bu seçenek, geliştirilecek çerçevenin, söz konusu arakatman gerçekleştirimlerinden bağımsız hale getirilmesini de gerektireceğinden, daha zor ve karmaşık olarak değerlendirilebilir. Ayrıca, verilen tanım alanı için istenen servis kalite değerleri de sağlanmayabilir.

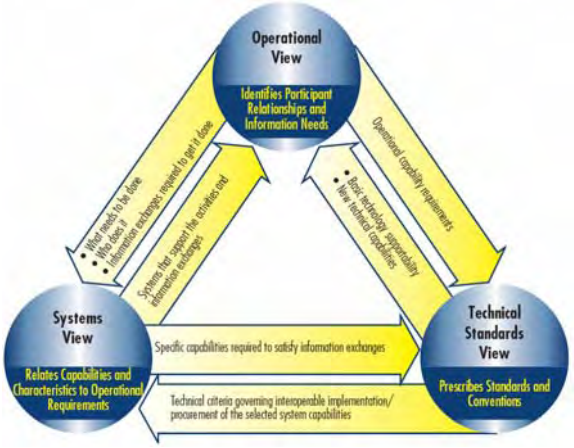
Sonuç olarak, bu makalede bahsedilen yazılım mimarisi geliştirme yaklaşımının yukarıda sözü edilen ikinci seçeneğe daha yakın olduğu söylenebilir. Dağıtık bir bilgi işleme sisteminde, farklı servis kalitesinde veri aktarımını destekleyen genel amaçlı bir yazılım çerçevesi geliştirmek öncelikli hedef olarak değerlendirilmektedir. Bunu, uygulama geliştirme altyapısı olarak kullanmak için, verilen bir tanım alanı için uyarlamak ve o alan için gerekli tüm servisleri sonradan eklemek gerekecektir.

2. Yazılım Mimari Standartlarına Uyumluluk

2.1. DoDAF

Genel geliştirme ve dokümantasyon ile ilgili olarak DoD (Department of Defense) Architecture Framework (DoDAF) [6] ile tanımlanan yaklaşımlar incelenerek uygulanması gerekli görülen kısımlar çerçeve standartları kapsamına alınacaktır. DoDAF, ilk olarak 1990'ların ortalarında bağlı ve çok ortaklı çalışmaları kolaylaştırmak amacıyla ortaya atılmış, daha sonraları da C^4ISR Mimari Çerçevesi (Architecture Framework) olarak uygulanmaya başlanmış, yapısal iletişim ve dokümantasyon standartlarını içeren UML'e uygun 3 ana görünüm altında 26 yapısal ürün tanımlayan bir araçtır. Proje kapsamında kullanılmak üzere ilk aşamada ürünler belirlenecek ve DoDAF'ye referans verilerek kullanılacaktır. Üst düzey süreçler böylelikle DoDAF'ye uyumlu olacaktır.

DoDAF teknik anlamda komut, servis ve aracı programcıkların farklı organizasyonlar arasında uyumlu çalışabilir olması ve gerek operasyonel, gerekse teknik olarak sistem düzeyinde uyumlu işleyişe sahip olmasını garanti etmek amacıyla tasarlanmıştır. DoDAF'nin sağlamış olduğu kurallar bütünü mimarilerin bir uyumluluk içinde geliştirilmesine olanak sağlar ancak gerçekleştirme detayları konusunda bir zorlamada bulunmaz. DoDAF'ın yapısal mimarisi Şekil 1'de verilmiştir.



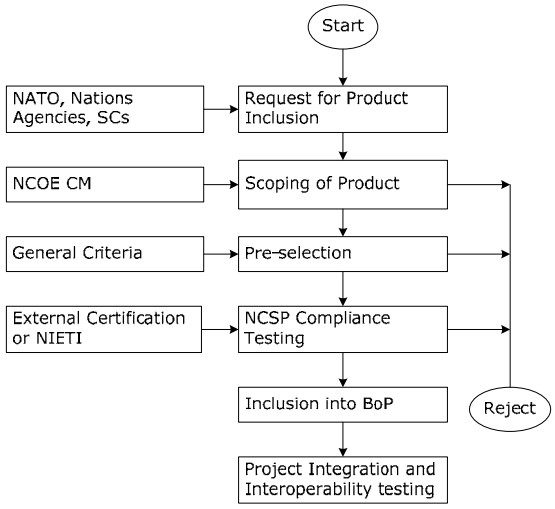
Şekil-1. DoDAF Yapısal Mimarisi

DoDAF dökümanlarında, DoDAF'ye uyumlu bir mimari çerçeve geliştirme sürecinde bulunması gereken minimum ürünler belirlenmiş ve uyumlu mimariler geliştirmek için bir modelleme sıralaması önerilmiştir. Bu makalede anlatılan yazılım mimarisi geliştirme projesinde de, öncelikli olarak minimum gerekli yapıtaşlarına ilgili DoDAF referansları verilecek; gerekli görülen hallerde diğer DoDAF ürünlerinden de faydalanılarak UML temeline dayanan üst düzey süreçler izlenilecektir.

2.2. NCOE

NATO Common Operating Environment (NCOE) [7], tüm NATO sistemleri için birlikte çalışabilirliği sağlamak amacıyla, sistemlerin yaşam döngüsünde kullanılan yazılım ürünlerini, fiyat ve uyumluluk faktörleri de dikkate alınarak tanımlayan bir NATO standartıdır. NCOE'nin ortaya çıkmasında temel amaç özellikle üretilecek NATO sistemlerinin aralarındaki (birlikte çalışabilme ile ilgili) uyumsuzlukları en aza indirmek ve böylece fiyat maliyetlerini düşürmektir.

Daha önceki yıllarda geliştirilen C^3 (İletişim, Komuta ve Kontrol; Communication, Command and Control) sistemlerinin çok değişik ürün yelpazesinde üretilmesi sonucu, ortaya çıkan ürünlerin kendi aralarında uyumsuzluklar ortaya çıkmıştır. Ayrıca, bir çoğunda kullanılan teknolojilerin yakın zamanda eskimesi ve desteğinin kalkması, büyük maliyet artışlarına yol açmaya başlamıştır. NCOE bu sorunları aşmak amacıyla ortaya atılmıştır. C^3 sistemlerinin hemen hepsi harita, veri güvenliği ve haberleşme gibi temel yazılım bileşenlerini içermektedir. Fakat, aynı işlevler ve yetenekler değişik sistemlerde farklı ve uyumsuz teknolojilerle geliştirilmiş ve sonuçta bu sistemler arasındaki birlikte çalışabilirlik (interoperability) özelliği ortadan kalkmıştır. Şekil 3'te NCOE ürün seçme süreci akış şeması gösterilmiştir.



Şekil-2. NCOE Ürün Seçme Süreci

NCOE'nin temel amaçları aşağıda listelenmiştir:

- Sistemlerin birlikte çalışabilirliğini sağlamak,
- Yaygın kullanılan yazılım servislerini NATO içinde ortak kullanıma almak, ki belirtilen standartlar NC3 Common Standards Profile (NCSP) da tanımlanmaktadır,
- Sistem geliştime zamanını, risk faktörlerini azaltmak ve ortak ihtiyaçlara göre uyumlu yada aynı ürünlerin kullanımını sağlamak,
- Sistem yönetim sürecinde, tasarım ve uygulama aşamalarında teknik eşgüdüme (coordination) ve maliyete katkıda bulunmak.

Sonuç olarak, NATO tarafından tanımlanan NCOE, tüm yazılım geliştirme, veri güvenliği, veri işleme/inceleme, işletim sistemi, ağ yönetimi, veri tabanları ve ofis uygulamaları gibi alanlarda uyumluluk ve birlikte çalışabilirlik açısından ortak bir yazılım bileşen kütüphanesi sağlamaktadır. NCOE, NATO ülkeleri içinde geliştirilen tüm sistelerde uyulması zorunlu bir standart olarak tanımlanmış değildir.

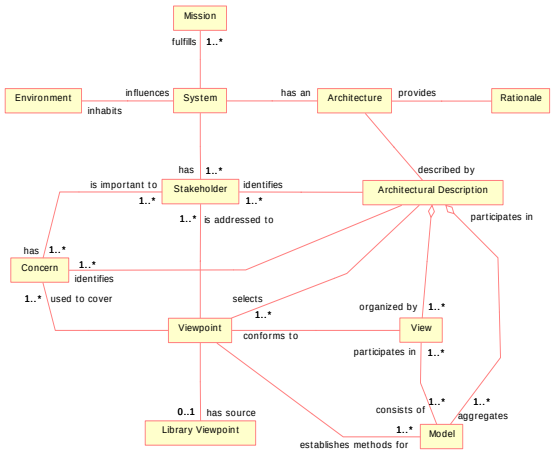
Bu makalede anlatılan mimari geliştirme projesindeki aşamalarda kullanılacak ürünlerin ve veri yapılarının NCOE'de tanımlı bileşenlerden seçilmiş olmasında büyük fayda olacağı düşünülmekte, ve kullanılan ve kullanılması muhtemel ürünlerde ilgili araştırmalar yapılacaktır.

2.3. IEEE Std 1471-2000

IEEE Std-1471-2000 [8], Ekim 2000'de yazılmış ve IEEE standartı olarak kullanıma sunulmuştur. Yazılım ağırlıklı (software intensive) karmaşık sistemlerde yazılım, tüm alt sistemin tasarım, inşa, kurulum, gelişme ve bakım aşamalarında çok önemli etkilere sahip olmaktadır. Bu nedenle, yazılım ağırlıklı sistemlerde başarılı bir geliştirme için, yazılım kritik bir eleman olarak karşımıza çıkmaktadır. Maliyet ve kalite açısından elde edilen avantajlar nedeniyle, mimari tanımlama (architectural description) yaklaşımının bu tür sistemlerde kullanımı giderek ağırlık kazanmaktadır.

IEEE Std-1471-2000 standardının temel amacı, tasarım sürecinin belirli standartlarda ve disiplinde yapılmasına yardımcı olmak ve bu şekilde yazılım ağırlıklı sistemlerde daha düşük maliyetli ve daha kaliteli sistemlerin üretilmesine olanak sağlamaktır. Bu yaklaşım, daha önceki yıllarda yapılan sadece donanım ağırlıklı mimarisel tanımlamalara ek olarak yazılımın da bu sistem tasarımı sürecinde entegre olarak düşünülmesi ve gerçekleştirilmesini öngörmektedir.

Mimari tanımlamanın kavramsal modeli, standartta tanımlandığı haliyle Şekil-3'te özetlenmektedir. Mimari yapı, bir sistemin içerdiği birimleri, aralarındaki ilişkileri, çevre ile etkileşimini tanımlama, girdi ve çıktılarını belirleme, temel tasarım kurallarını somutlaştırma işidir. Mimari tanımlama ise bir mimariyi dokümante etmek için hazırlanan ürünlerin toplamıdır.



Şekil-3. IEEE Std 1471-2000 Mimari Tanımlamasının Kavramsal Modeli

Sistem, önceden tanımlanmış bir veya birden çok işlevi yerine getirmek üzere entegre olmuş değişik birimlerden oluşan bir yapıdır. Standart açısından bakıldığında bu tanımlama temel teşkil etmektedir. Bir sistem, alt sistemlere sahip çok karmaşık bir yapıda olabilir veya çok temel alt birimlerin birbiriyle uyumlu bir şekilde tümleştirilmesi sonucu basit bir sistem de olabilir. Sistem, bir ortam içinde bulunur. Bu sistemi çevreleyen ortam, değişik şekillerde sistemi etkilemektedir. Çünkü bu ortamda çok farklı sistemler bulunmakta ve bunlar da kendi aralarında bir etkileşim içinde olabilmektedirler. Fakat ortam, sistemin etkileşim sınırlarını belirlemektedir.

Her sistemin bir veya birden çok paydaşları/sahipleri (stakeholder) bulunmaktadır. Her paydaşın da yine bir veya daha çok ilgilileri (concern) vardır. Bu ilgililer sistem gelişimine bir şekilde katkıda bulunurlar. Her sistem, bir görevi yerine getirmek için çalışmaktadır. Çünkü her sistemin bir var olma amacı bulunmaktadır. Genelde sistemler paydaşlarının birtakım amaçları doğrultusunda çalışmaktadırlar.

Yukarıda tanımlanan terminolojiler daha çok sistem ve bulunduğu çevre ile ilgilidir. Fakat IEEE Std 1471-2000 mimarisel tanımlama anlamında standartlar sunmaktadır. Bu anlamda aşağıdaki terminolojiler bu standartla ilgilidir: Her sistem bir mimariye sahiptir ve bu mimari tarif edilebilir anlaşılır bir formata dönüştürülebilir. Sistem mimarisi kavramsaldır, fakat bu mimarinin tanımlanması somut bir olaydır. Mimariler bir mantıksal açıklama (rationale) sunarlar. Özet anlamıyla mimarisel tanımlama, mimariyi dokümante etmek için gerekli olan ürünlerin toplamı olarak tanımlanabilir. Bu mimarisel tanımlama bir veya birden çok görünüme (view) bölünebilir. Görünümler, sistemin değişik bakış açılarından tasviridir. Her görünüm, bir veya daha çok paydaşın ilgi alanını kapsamaktadır. Görünümler, bakış açısında (viewpoint) tanımlanan kurallara göre oluşturulur. Bakış açıları, görünümleri oluşturmak ve kullanmak için gerekli olan birtakım geleneksel tarifleri tanımlarlar. Her görünüm, bir veya daha çok modelden oluşabilir. Aynı zamanda her model de bir veya daha çok görünüme ait olabilir. Modellerin oluşturulma işlemi de bakış açılarında tanımlanan kurallara göre yapılır. Bakış açıları, başka bir yerde tanımlanmış da olabilir. Bu şekilde tanımlanan bakış açıları, kütüphane bakış açıları olarak adlandırılırlar.

Mimari tanımlama aşağıdaki adımları içermektedir:

- Sistemin ve gelişiminin tanımlanması,
- Sistem pay sahipleri ile haberleşme ve eşgüdümünün sağlanması
- Değişik mimari yapıların karşılaştırılması ve değerlendirilmesi,
- Sistem geliştirme aktivitelerinin planlanması,
- Olası işlevsel değişikliklerin belirlenmesi,
- Sistemin doğruluğunun mimari tanımlamalar kullanılarak kanıtlanması,
- Elde edilen bilgilerin standarta göre dokümante edilmesi.

Bu makalede bahsedilen yazılım mimarisi geliştirme projesi kapsamında IEEE Std 1471-2000 standardının Mimari Tanımlama yöntembiliminin uygulanabilirliği araştırılmakta ve uygulama modellemelerini üzerine de çalışmalar planlanmaktadır.

3. DDS (Data Distribution Service) Tabanlı Mimari Tasarımı

3.1. DDS ile ilgili Temel Bilgiler

OMG (Object Management Group), görevi, arakatman standartlarını belirlemek olan uluslararası bir organizasyondur. OMG, son olarak, gerçek zamanlı sistemlerde, yayın/abone (publish/subscribe) iletişimi için standart bir arayüz olan *DDS (Data Distribution Service)* belirtimini (specification) [5] benimsemiştir. Genel tanımıyla DDS, servis kalitesi (Quality of Service, QoS) farkındalığında, yayın/abone mantığına sahip bir arakatman altyapı standartıdır. DDS'in temel mantığı, bilgi değiş tokuşu değil, bilgiye erişimdir. Verinin tutulduğu paylaşımlı bölge (örneğin paylaşımlı bellek ya da shared memory) mimarisine sahiptir, konu-tabanlı (topic-based), içerik-tabanlı (content-based) ve tür-tabanlı (type-based) yayın/abone haberleşmesini sağlamaya dayanır. DDS, heterojen bilgisayar ağları, dinamik ortamlar ve güvenilir olmayan taşıma (transport) katmanları için uygundur.

Bu makalede anlatılan yazılım mimarisi geliştirme projesi, veri merkezli bir mantıkla çalışan komuta kontrol tanım alanı uygulamalarını destekleyecektir. Veri merkezli sistemlerin şu özellikleri vurgulanmaktadır:

- Veri alışverişinde bulunan katılımcılar dağıtık bir yapıdadır,
- Katılımcılar arasında nesne değil, veri akışı mevcuttur,
- Akan verinin çeşitli servis kalitesi değerlerine sahip olması gereklidir,
- Veri üzerindeki işlemlerde bir zaman kısıtlaması bulunmaktadır.

3.2. Arakatman Mimarileri ve DDS

Arakatman mimarileri denince temelde iki bilinen yöntem akla gelir: *İstemci/Sunucu* (dosya sistemleri, veri tabanları, CORBA, DCOM) ve *Peer-to-Peer***.** (telefon, TCP). İlki, eğer veri, doğası gereği merkezi ise daha yararlıdır, başarımda darboğaz (bottleneck) kaçınılmazdır. İkincisinde ise yüksek bant genişliğini basit modellemelerle elde etmek mümkündür.

Veri dağılımı ve verinin yönetilmesi açısından durum ele alındığında ise dağıtık veri tabanlarındaki *kopyalanmış veri (replicated data)* ve *yayıncı/abone mesajlaşması (publish/subscribe messaging)* değerlendirilmektedir. Bu belirtilen farklı bakış açılarının tümünün avantajlı veya dezavantajlı oldukları yanlar olmakla birlikte geliştirme süreci açısından da temel farklı bakış açıları bulunmaktadır.

Temelde eldeki verinin doğası gereği merkezi bir yerden mi yönetilmesi gerektiği yoksa dağıtık bir yapıya mı sahip olduğu öncelikli olarak incelenmeli, ve bu inceleme sonucunda edinilen görüşlere göre bir analiz ve geliştirme süreci tercih edilmelidir. DDS tabanlı bir mimarinin en belirgin özelliği veri iletiminde daha verimli olabilmesidir. Bu verimlilik gerek yayıncının, gerekse abonenin veri aldığı ya da gönderdiği yer ile ilgili bilgi sahibi olmasına ve bu bilgiyi sağlamasına gerek olmayışıdır. Verinin sadece gönderildiği ya da alındığı yer ortaklaşa yönetilen paylaşımlı bir alandır (ortak kullanılan veri alanı, dağıtık veritabanı, ortak kullanılan bellek, v.b.). Bu özelliğin bir sonucu olarak gönderilen veriye karşılık herhangi bir yanıt (ya da onay) beklenmiyor olması da kabul edilen zaman kaybını minimuma indirgemektedir.

DDS tabanlı çerçeve mimarisinin bir diğer önemli avantajı, veri üzerine filtreleme gibi de düşünülebilecek servis kalitesi parametrelerine sahip olmasıdır. Bu şekilde her iletilen veri parçası üzerindeki kararlılık ve güvenilirlik daha fazla olmakta, bant genişliği daha verimli düzeylerde tutulmaktadır. Buna paralel olarak diğer çerçeve mimarilerinde önemli verimlilik kaybına neden olabilen çevrimli (cyclic) mesajlardan kaçınılması da DDS'in çevrimsiz (acyclic) mesajlaşma temeline dayanması nedeniyle mümkün olmaktadır.

DDS tabanlı çerçeve mimarilerinin bir başka özelliği de daha gevşek eşleme olarak da değerlendirilebilecek bir biçimde verinin her istenilen parçası için ayrı bir istek yapmak yerine, belirli bir konuya bağlı olarak veri üzerine genel bir istek bildirmektir. Bu mimari yapısı, geliştirme sürecinde kodlamayı da kolaylaştırmakta, uygulamaların sadece veri alıp verme metodlarını barındırmalarını yeterli hale getirmektedir.

Yukarıda değinilen temel noktalar ışığında DDS ile CORBA, özetle aşağıdaki maddeler şeklinde karşılaştırılabilir:

- DDS, veri broker, CORBA ise nesne broker'dir.
- DDS, veri-merkezli yayın/abone mantığına dayalıdır. CORBA, nesne-merkezli istemci/sunucu mantığına dayalıdır.
- DDS'te, bileşenler arasında zayıf bağlaşım (coupling) bulunur. CORBA'da, bileşenler arasında sıkı bağlaşım bulunur.
- Veri, serbest bırakılmadıkça DDS'in kontrolündedir. CORBA'da, veri, çağrılma (invocation) sonrası istemci tarafından sahiplenilir.
- Her ana bilgisayar (host) bir DDS anına sahiptir. CORBA'da, her işlem, bir ORB anına sahiptir.

DDS, son yıllarda oldukça yaygınlaşan bir standart haline gelmiştir. Çeşitli firmalar, bu standardın gerektirdiği şartları uygulama olarak da geliştirmiş ve ticari ürün haline de getirmişlerdir. Hem DDS hem de CORBA'nın altyapısı olarak görülebilecek ACE [9] ve TAO [10] ise ücretsiz ve açık kaynaklıdır.

3.3. Temel DDS Özellikleri ve Bunların Performansla İlişkisi

DDS'in temel özelliklerinin, geliştirilen veri merkezli yapıya uygulanabilirliği ve bu özelliklerin başarım açısından sağlayacağı katkı, maddeler halinde incelenmiştir:

- **QoS Parametreleri:** Konu (topic) üzerinde tanımlanan QoS parametreleri sayesinde ne kadar sıklıkla veri yayınlanacağı, bu verinin ne kadar yaşayacağı, v.b. gibi özellikler belirlenerek servis kalitesi ayarlanabilir ve başarım artırılabilir. Örneğin, yukarıda veri merkezli sistemler için sözü edilen zaman kısıtı, QoS parametreleri ile sağlanabilir.
- **Veri Akışı:** DDS mimarisinde, konuyu yayınlayan yayıncılar (publishers) hedef belirtmezler. Konuyla ilgili aboneler (subscribers), eğer konu tipi ve QoS'u tutuyor ise, global veri alanındaki veriye kendileri erişirler. Yani veri akışı DDS arakatmanı tarafından otomatik olarak kontrol edilir. Böylece bu arakatman sayesinde, daha yüksek bir başarım sağlayan aracısız bir bağlantı kurulmuş olur.
- **Kullanıcı Tanımlı Veri Tipleri:** Önceden tanımlı veri tiplerinin içini doldurmak yerine, sadece gerekli alanlara sahip yeni veri tipleri tanımlayarak daha verimli bir iletişim sağlanır.
- **Otomatik Keşif:** Arakatman sayesinde katılımcılar birbirlerini otomatik olarak bulabilirler. Böylece ortama yeni dahil olan ya da değişikliğe uğrayan katılımcılar, fazladan bir işlem gerektirmeden iletişime başlayabilirler. Dolayısıyla mimari, belirli bir ölçülebilirlik kazanır.
- **Veri Filtreleme ve Dönüştürme:** DDS arakatmanı, konuları içeriklerine göre filtreleme ve dönüştürme işini üstüne aldığından, DDS ile yazılacak uygulamalarda veri işleme işini kolaylaştırarak başarımı artırır.
- **Veri Dağıtma:** Gerekli olduğunda QoS parametreleri ile veri dağıtım tipi değiştirilebilir. Örneğin, hatasız iletişimin çok önemli olduğu durumlarda yeni veri göndermek için karşıdan onay (acknowledgement) beklenirken, hızın önemli olduğu durumlarda onay beklenmeyebilir.

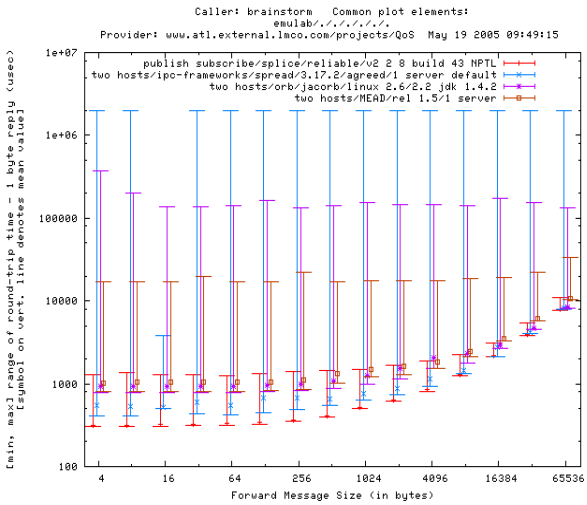
Herhangi bir arakatmanın performansını ölçmek için başvurulabilecek iki yol ve bu konularda DDS'in sağladıkları şu şekilde verilebilir:

1. Yayıncıdan alıcıya veri transferi süresi:

- Onay beklenmeyen durumlarda transfer süresi kısalarak performans artar.
- Kullanıcı tanımlı veri tipleri sayesinde gereksiz alanlar atılarak transfer süresi azalır.
- Kullanıcı tanımlı veri tipleri sayesinde kullanıcı ve arakatman veri tipi gibi iki ayrı veri tipi arasında dönüşüm yapma gereği ortadan kalkarak transfer süresi azalır.
- Yayıncı ve abone arasında aracısız iletişim kurulduğu için transfer süresi azalır.
- Aboneler "sıfır kopyalama (zero copy)" özelliği sayesinde global veri alanındaki veriye, kendilerine kopyalamadan ulaşabilirler. Bu da tranfer süresini azaltır.

2. Birim zamanda yayıncıdan alıcıya akan veri miktarı:

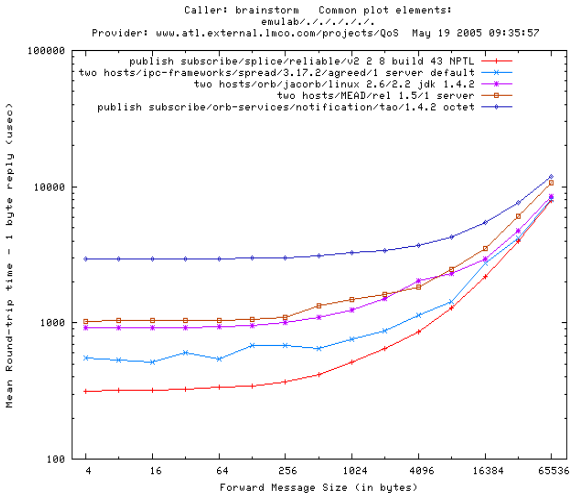
- Kullanıcı tanımlı veri tipleri sayesinde gereksiz alanlar atılarak birim zamanda akan veri miktarı artar.
- Data filtreleme sayesinde data işleme hızı arttığından birim zamanda akan data miktarı artar.
- Aracısız iletişim sayesinde kaynak yetersizliği sorunları azalacağından birim zamanda akan veri miktarı artar.



Şekil-4. OpenSplice DDS ile çeşitli CORBA gerçekleştirimlerinin ortalama veri değiş-tokuş hızlarının karşılaştırılması

DDS gerçekleştirimi olarak projemiz kapsamında deneme sürecinde kullanılmış olan *Open Splice DDS*'in [11] veri gidiş-geliş hızıyla (round-trip time) ilgili *LMCO-ATL*'den [12] alınan bazı grafikler aşağıda sunulmaktadır. Şekil-4'te Open Splice DDS ile çeşitli CORBA gerçekleştirimlerinin ortalama veri gidiş-geliş hızlarının karşılaştırılması verilmektedir, ve görüldüğü üzere, bütün durumlarda DDS performansı daha yüksektir. Şekil-5'te ise Splice DDS ve çeşitli CORBA gerçekleştirimlerinin maksimum ve minimum veri gidiş-geliş hızlarının karşılaştırılması yapılmaktadır. Bu grafikten çıkarılan önemli sonuç, maksimum ve minimum veri gidiş-geliş hızı arasındaki fark en az DDS'de

olduğu gibi en uygun ortalama gidiş-geliş hızı da DDS'dedir.



Şekil-5. OpenSplice DDS ve çeşitli CORBA gerçekleştirimlerinin maksimum ve minimum veri değiş-tokuş hızlarının karşılaştırılması

DDS sağladığı aşağıdaki özelliklerle *komuta-kontrol* sistemlerine uygunluğunu kanıtlamaktadır:

- Az gecikme sağlayan, "best effort" (aboneden onay beklememe durumu) dağıtım mekanizması,
- Servis kalitesini ve kaynak kullanımını belirleyen QoS parametreleri,
- İletişim ve bağlantı hakkında durum bildirimleri,
- Yukarıda belirtilen düşük transfer zamanı ve birim zamanda yüksek miktarda veri akışı özellikleri.

3.4. DDS Tabanlı Ürün Gereksinimleri

DDS in veri merkezli yapısından dolayı geliştirilecek sistemin ve alanının veri analizinin yapılmış olması gerekir. Bu analiz gelinen aşamada minimum düzeyde gerçekleştirilebilmiştir. İleride bu veri analizi için de kapsamlı bir sistem mühendisliği çalışması gerekliliği dahilinde planlamalar yapılmaktadır.

Yayın/Abone sistemlerinin karakteristik özellikleri: *Sürerlik (Persistence), İşlembilgisel Garantiler (Transactional Guarantees), Sıralama (Ordering), Hızlılık (Urgency), Öncelikler (Priorities)* olarak sıralanabilir. Bu işlemler kullanılan DDS gerçekleştirimi tarafından sağlanmaktadır. Çerçeve, bu DDS fonksiyonlarına kullanıcının erişimini düzenlemek ve kolaylaştırmak ile alan gereksinimlerine göre gerçekleştirmek amacında olmalıdır.

Temel Servis kalitesi (QoS) işlemleri:

- Servis Kalitesi Anlaşması (QoS Negotiation)
- Servis Kalitesi Uyumu (QoS Adaptation)
- Servis Kalitesi İzlenmesi (QoS Monitoring)

Bu operasyonlar içerisinde servis kalitesi anlaşması ve servis kalitesi uyumu DDS tarafından sağlanmaktadır. Servis kalitesi ise çerçeveye dahil edilecek bir *sistem kontrol ve izleme modülü* tarafından izlenebilmelidir. Bu modül, 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.

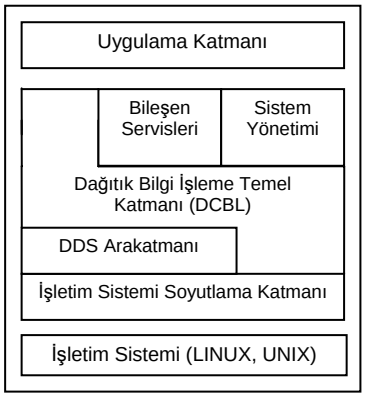
4. Komuta-Kontrol Sistemleri için Ulusal Yazılım Mimarisi (HVL_FRWK)

HAVELSAN'ın daha önce geliştirdiği bir çok projede yazılım geliştirme altyapısı olarak kullandığı uygulama çerçeveleri, önceden sağlanmış hazır katmanların tümleştirilmesi ile oluşmuş çerçevelerdir. Yazılım geliştiriciler, ACE, CORBA, v.b. gibi çerçeve/arakatmanların üzerinde inşa edilmiş servislerin kullanıldığı özel çerçeveler üzerinde uygulama geliştirerek, oldukça önemli bir bilgi birikimi ve tecrübe edinmişlerdir. Ancak, sadece hazır sunulan servisler kullanılarak yazılım geliştirmeyle yetinilmemeli, komuta kontrol projelerinde, HAVELSAN tarafından üretilmiş, yeniden kullanılabilir, güvenilir ve taşınabilir bir yazılım mimari altyapısı oluşturulmalıdır. Bu amaçla, ACE ve DDS gerçekleştirimi üzerine inşa edilecek katmanlı bir yazılım mimarisi esas alınmıştır. *HVL_FRWK* ulusal yazılım mimarisi, Şekil-6'da verilmiştir.

HVL_FRWK geliştiriminin sağlayacağı yararlar şu şekilde özetlenebilir:

- Uygulama geliştiricilerini pek çok sistem karmaşıklığından uzak tutar,
- Veriye dayalı ve olay güdümlü tasarım sağlar,
- Bileşenler yerine veriye bağlı basit tasarım sağlar,
- Uygulamalar pek çok servis detayından uzaktır (Kayıt-Yeniden Oynatma (Record&Playback), Denetim Noktası-Kurtarma (Checkpoint&Recovery), v.b.),
- Uygulama gerekli veriyi sağlar ve işleme için gerekli veriyi alır,
- Veri sanal evrensel paylaşılan bellekte konular üzerinden yönetilir,

- Görüntüleme sistemleri tasarım ve gerçeklemesi daha basittir (veri depolarına gerek yoktur).



Şekil-6. HVL_FRWK Komuta-Kontrol Sistemleri için Ulusal Yazılım Mimarisi

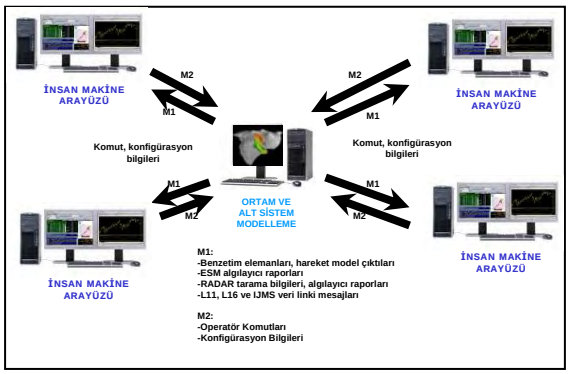
Bu yararlar yanında, aşağıdaki zorluklar da göz önüne alınmalıdır:

- Yeni geliştirme yöntembilimine (methodology) gerek duyulmaktadır,
- Geliştirme istemci/sunucu mimarisinden farklı bir bakış açısıyla yapılmalıdır,
- Halihazırda DDS hakkında yeterli geliştirme deneyimi mevcut değildir.

4.1. DDS Tabanlı Yazılım Mimarisi üzerine Prototip-1 Gerçekleştirimi

HAVELSAN tarafından geliştirilen gerçek zamanlı ACE ve DDS tabanlı arakatman yazılımı, mimarinin performasını gösterebilmek amacıyla çeşitli basit komuta-kontrol uygulamalarıyla test edilmektedir. *Prototip-1* olarak ise genel anlamda bir arama/kurtarma gemisinin temel seyir, arama/kurtarma, helikopter kontrol ve yönlendirme komutlarının görev arayüzlerinden sağlandığı ve ortam/altsistem modellemesinin bir sunucudan yapıldığı bir uygulama bu makalede sunulmaktadır. Prototip-1 ana mimarisi Şekil-7'de gösterilmektedir.

Prototip-1 gerçekleştirimi Şekil 6'da da değinilen katman mimarisine göre analiz edildiğinde İşletim Sistemi olarak LINUX, İşletim Sistemi Soyutlama Katmanı olarak ACE, ve bu katmanların üzerinde dağıtık veri iletişimini sağlamak amacıyla TAODDS [13] açık kaynaklı DDS implementasyonu, ve OpenGL yeteneklerinden faydalanılarak geliştirilmiş 3 boyutlu gerçek dünya koordinatlarına göre ortam gösterimi yapabilen ve komut girişine imkan veren bir kullanıcı arayüzünden oluşmaktadır. Bu operatör arayüzü Şekil 6'de gösterilmiştir. Daha önceki denemelerimizde OpenSplice DDS'i de kullanmamıza rağmen, açık kaynaklı olması nedeniyle bu prototipte TAODDS tercih edilmiştir.



Şekil-7. HVL_FRWK Prototip-1 Ana Mimarisi

Ortam ve Altsistem Modelleme kapsamında arama/kurtarma gemisi seyir işlemleri, diğer unsurların (dost ve düşman gemileri, uçaklar, helikopterler) kinematiği ve davranışları, radar, taktik veri linki ve baş top modellenmiş ve simülasyon ortamı oluşturulmuştur. Farklı bilgisayarlarda çalışan simülatörler, görev konsolları ve ana bilgisayar arasındaki gerçek zamanlı, yüksek hacim ve süratli veri iletişimi DDS tabanlı arakatman yazılımıyla sağlanmaktadır.

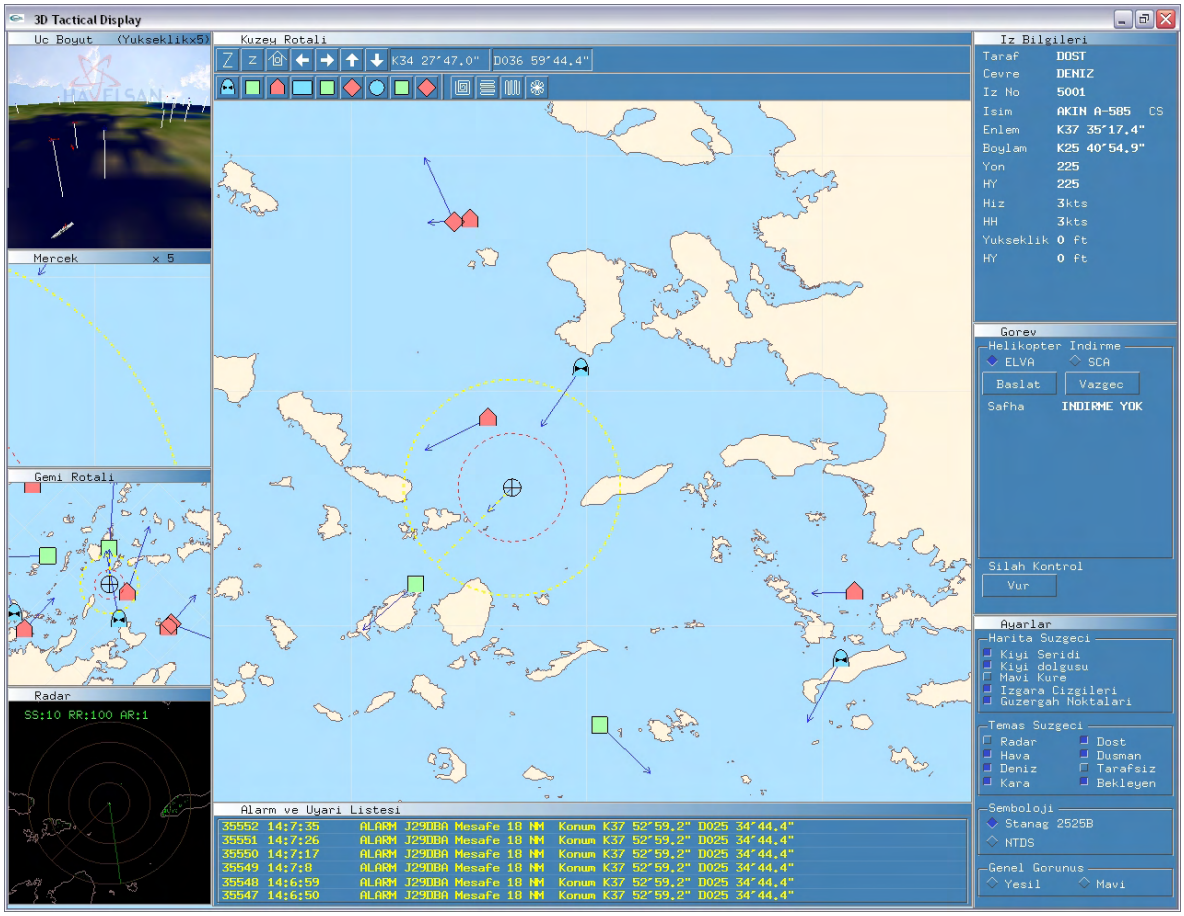
Prototip-1 uygulamasını oluşturan yazılımlarla DDS tabanlı bir arakatmanın komuta kontrol uygulamalarına uygunluğu hem kavramsal hem de teknik olarak doğrulanmıştır. Bu konuda performans ölçüm ve karşılaştırma çalışmaları devam etmektedir, ancak ilk edinilen izlenimler oldukça umut vericidir.

5. Değerlendirme ve Sonuç

HAVELSAN A.Ş. bünyesinde geliştirilmeye başlanan komuta-kontrol sistemleri için ulusal yazılım mimarisi geliştirme projesi, temel amacına da paralel olarak bu alandaki uluslararası kabul görmüş standartlara uygun ve yeni teknolojileri barındıran bir arakatman geliştirme projesidir.

Bu makalede değinilen bu standart ve teknolojilerin komuta-kontrol uygulamalarında gereksinimler doğrultusunda veri analizleri ve kaynak planlamalarına paralel olarak geliştirilmesi öngörülmektedir.

Makalenin sonlarında anlatılan deneme uygulama çalışması Prototip-1, yukarıda bahsedilen amaç, hedef ve öngörülerin gerçek anlamda bir komuta-kontrol uygulamasında kullanılabilirliğini ortaya koymuştur. HVL_FRWK ulusal yazılım mimari geliştirme projesi, gerek geliştiricilerin gerekse kullanıcıların faydalanabilecekleri, bu alanda geliştirilecek proje ve uygulamaların kaynaklarının daha verimli kullanılabileceği bir yapı sunmaktadır. Komuta-kontrol uygulamalarında önemli bir yeri olan yüksek ölçekli veri iletişiminin de yüksek hızlarda başarılabilmesi ve performans testlerinin ilk izlenimler ışığında umut verici olması



Şekil-8. HVL_FRWK Prototip-1 Operatör Komuta-Kontrol Arayüzü

geliştirme projesinin temel amaç ve hedeflerini haklı çıkarmada katkı sağlamaktadır.

HAVELSAN A.Ş. olarak ulusal yazılım mimarisi geliştirme projesi çalışmaları öngörülen hedefler doğrultusunda devam etmekte; ve geliştirici ve kullanıcılara rahatlıkla yararlanabilecekleri ulusal bir yapı sunma hedefine emin adımlarla ilerlemektedir.

6. Kaynakça

[1]. ObjectWeb Konsorsiyum Web Sitesi, http://www.objectweb.org/.

[2]. OMG'nin CORBA Web sitesi, http://www.corba.org/.

[3]. Microsoft DCOM (Distributed Component Object Model), http://www.microsoft.com/com/.

[4]. JAVA RMI (Remote Method Invocation), http://java.sun.com/products/jdk/rmi/.

[5]. OMG'nin Data Distribution Service (DDS) v1.0 spesifikasyonu, http://www.omg.org/docs/formal/04-12-02.pdf.

[6]. C. Kobryn and C. Sibbald, Modeling DoDAF Compliant Architectures: The Telelogic Approach for Complying with the DoD Architectural Framework, 2004, http://www.uml-forum.com/docs/papers/White_Paper_Modeling_DoDAF_UML2.pdf.

[7]. NATO C3 Tecnical Arcitecture, NC3TA Vol. 1-5, http://194.7.80.153/website/home_volumes.asp?menuid=15.

[8]. IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems – Description, http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html.

[9]. ACE (The ADAPTIVE Communication Environment) Web Sitesi, http://www.cs.wustl.edu/~schmidt/ACE.html

[10]. TAO (The ACE Orb) Web Sitesi, http://www.cs.wustl.edu/~schmidt/TAO.html

[11]. PrismTech'in OpenSplice DDS Web Sitesi, http://www.prismtechnologies.com/section-item.asp?id=557&sid=18&sid2=10&sid3=252

[12]. Lockheed Martin Corp., Advanced Technology Labs' Agent and Distributed Objects Quality of Service (QoS) Web Sitesi, http://www.atl.lmco.com/projects/QoS/

[13]. Object Computing Inc.'in TAODDS Web Sitesi, http://www.ociweb.com/products/dds

SİNYAL İŞLEME YAZILIMLARI İÇİN MİMARİ TASARIMI

Dinç ACAR
ASELSAN A.Ş.
dacar@mst.aselsan.com.tr

Sevda ERDOĞDU
ASELSAN A.Ş.
erdogdu@mst.aselsan.com.tr

Alaattin DÖKMEN
ASELSAN A.Ş.
dokmen@mst.aselsan.com.tr

Metin ŞENGÜL
ASELSAN A.Ş.
sengul@mst.aselsan.com.tr

Mustafa YAMAN
ASELSAN A.Ş.
yaman@mst.aselsan.com.tr

Öz

Sinyal işleme yazılım geliştirme çalışmalarında genelde performans kriterleri ön plandadır ve yazılımlar performansa göre eniyilenir. Ancak performansa göre eniyilenmiş sinyal işleme yazılımlarında yeni gerekler oluştuğunda ya da benzer projelerde kullanılmak istendiğinde zorluklar yaşanır. ASELSAN'da son zamanlarda birbirlerine benzer projelerin sayısının artmasıyla ve proje sürelerinin kısalmasıyla ortak mimari kullanımı gerekliliği ortaya çıkmıştır. Sinyal işleme yazılımları için bu aşamada önemli olan sadece performans değil, bunun yanında belirlenen kalite kriterlerine uygunluğu, geliştirme maliyetleri ve yazılım tasarımında değişikliğe izin veren esnekliğin bulunmasıdır. Bu bildiride Sinyal İşleme Yazılımları Referans Mimarisi geliştirmede izlenen yol ve sonuç olarak geliştirilen mimari sunulmaktadır.

1. Giriş

ASELSAN Mikrodalga ve Sistem Teknolojileri (MST) Grubu'nda Radar, Elektronik Harp ve Silah Sistemleri projelerinde kullanılmak üzere sinyal işleme (Sİ) yazılımları geliştirilmektedir. Yakın zamana kadar proje gereklerindeki çeşitlilik ve geliştirme zamanlarının farklılığından dolayı Sİ yazılımları için ortak bir yaklaşıma gidilmezken, zaman içinde proje sayısının ve projelerin arasındaki benzerliklerin artmasıyla ortak mimari yapı oluşturulması gündeme gelmiştir. Bunun üzerine bu bildiride belirtilen Sİ Yazılımları için Referans Mimari belirleme çalışmaları gerçekleştirilmiştir.

Bugüne kadar Sİ yazılımlarının geliştirilmesi çalışmalarında yalnızca performans kriterleri ön planda tutulmuştur. Ancak, proje ihtiyaçları doğrultusunda Sİ yazılım mimarilerinin ortaklanması gereğiyle birlikte, performans kriterlerine ek olarak, belirlenen kalite faktörlerine uygun, yazılım geliştirme maliyetlerini düşürecek ve yazılımda değişikliğe izin veren bir yazılım mimarisi tasarlanması yoluna gidilmiştir.

Mimarinin oluşturulması için öncelikle gerekleri belirlemek amacıyla alan ve ürün analizi yapılmıştır. Bu kapsamda ASELSAN MST Grubu'nda tamamlanıp teslim edilmiş ve devam etmekte olan projelerdeki Sİ yazılımlarının özellikleri ve donanımlarla olan ilişkileri incelenmiş ve ortak gerekler belirlenmiştir. Bu gereklerden ve kalite faktörlerinden yola çıkarak mimari gereksinimler belirlenmiş ve Sİ Yazılımları Referans Mimarisi oluşturulmuştur. Son olarak, oluşturulan yazılım mimarisinin değerlendirilmesi için örnek senaryolardan faydalanılan bir yöntem uygulanmıştır.

Bu bildiride ilk olarak mimari gereklerin belirlenmesi çalışmaları anlatılacaktır. Ardından oluşturulan mimari tanıtılacak ve sonrasında mimarinin değerlendirilmesi yapılacaktır.

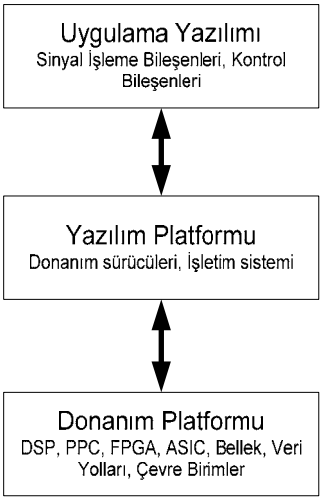
2. Mimari Gereklerin Belirlenmesi

2.1. Alan ve Ürün Analizi

ASELSAN MST Grubu'nda geliştirilen sistemlerde genel olarak grafiksel kullanıcı arayüzünü sağlayan yazılımlar, RF / görüntü / ses sinyallerini işleyen sinyal işleme yazılımları ve sistem birimlerini kontrol eden ve senaryoları yöneten kontrol işlemcisi yazılımları bulunur.

ASELSAN MST grubunda geliştirilmiş ve teslim edilmiş projeler Referans Yazılım Mimarisi geliştirme çalışmasına baz alınmıştır. Geçmiş projelerin ve devam etmekte olan projelerin Sİ yazılım tasarımları yazılım-donanım-algoritma üçgeninde incelenmiştir.

Projeler incelendiğinde her birinde, uygulama yazılımı, yazılım platformu ve donanım platformu bileşenlerinin bulunduğu görülmektedir.



Şekil 1: Projelerdeki ortak sinyal işleme bileşenleri

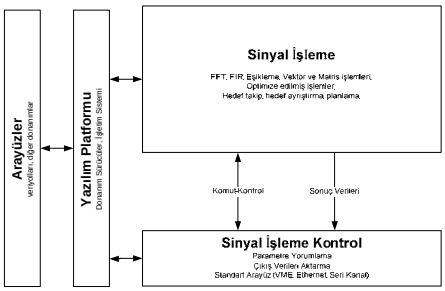
Uygulama yazılımı iki bölümden oluşmaktadır: Sinyal İşleme Bileşenleri ve Kontrol Bileşenleri. Sinyal İşleme Bileşenleri, filtreler, kodlama blokları, algoritmalar gibi sinyal işleme ile ilgili tüm işlevleri kapsamaktadır. Kontrol Bileşenleri, sinyal işleme bloklarının üst seviye kontrolünü, diğer işlemcilerle haberleşmeyi, komut-kontrol arayüzlerini kapsamaktadır.

Yazılım platformu, uygulama yazılımı ile donanım platformu arasındaki köprüyü oluşturmaktadır. Bu bölüme, yazılımın iletişim kurduğu tüm donanımların sürücüleri ile eğer kullanılıyorsa işletim sistemi girmektedir.

Donanım platformu, yazılımın kullandığı tüm donanımlardan oluşmaktadır. Donanım platformunda DSP işlemcileri, FPGA ve ASIC yongaları, bellek, veri yolları ve diğer çevre birimler bulunmaktadır.

Projelerdeki sinyal işleme yapılarındaki bloklar daha detaylı incelendiğinde Şekil 2'de yer alan şema hazırlanmıştır.

Şekil 2'de görülen Arayüzler kutusu, kontrol işlemcisi ve test birimleri ile haberleşmek için kullanılan standart yapılardaki veri ve kontrol arayüzünü ve işlenecek sinyalin işlemciye akışının sağlandığı ve diğer donanım birimleri ile haberleşen ve çoğu zaman hızlı veri akışını destekleyen, projeye özel hızlı arayüzü kapsamaktadır. Şekil 1'de yer alan "Uygulama Yazılımı" bloğu ise Şekil 2'de ikiye bölünmüştür: Sinyal İşleme ve Sinyal İşleme Kontrol.



Şekil 2: Sinyal işleme yazılımlarında arayüzler

2.2. Sinyal İşleme Yazılımı Geliştirme Çalışmalarında İzlenen Yöntem

Projelerde Sİ yazılım geliştirme faaliyetlerine başlandığında genelde izlenen yöntem Şekil 3'te gösterildiği gibidir. Çalışmaya, kullanılması hedeflenen donanımlar üzerinde bazı temel süre ölçümlerini alma ile başlamak, algoritma alternatiflerinin değerlendirilmesinde ve donanım-yazılım ayrıştırmasında kolaylık sağlayacaktır. Bu çalışmanın ardından gerçekleştirilecek algoritma tasarımının ilk aşamada ana akışları ve kabaca işlem yüklerini göstermesi yeterli olacaktır.

Bir sonraki aşama, algoritmaya göre işlevlerin yazılıma ve donanıma atanmasıdır. Bu işlem projede bulunan tecrübeli mühendisler tarafından yapılmaktadır.

Ayrıştırma yapıldıktan sonra yazılım ve donanım paralelden geliştirme çalışmalarına başlayabilmektedir. Bu aşamadaki çalışmaları yazılım, donanım ve algoritma mühendisleri koordineli olarak yürütmektedirler.

Yazılım ve donanım çalışmaları yürütülürken algoritma mühendislerine işlem güçleri ve bellek kullanımı konusunda geri beslemeler verilmektedir.

Yazılım mühendisleri tasarımlarını oturtabilmek için donanımın olgunlaşmasını beklemek durumda kalmaktadır.

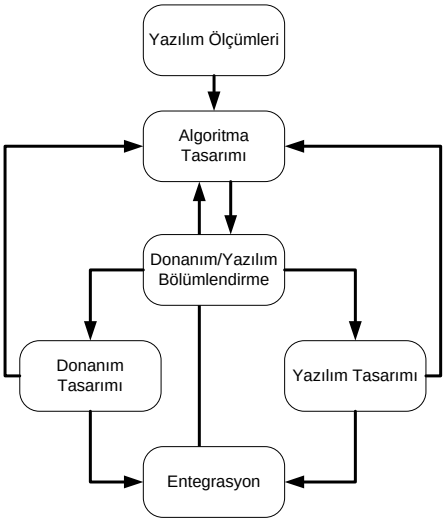
Yazılım ve donanımlar belirli olgunluk seviyesine geldiklerinde entegrasyon çalışmaları başlatılmaktadır.

Gelişen teknoloji ile FPGA kullanımının daha verimli olması özellikle akan veriler üzerindeki işlemlerin FPGA üzerinde gerçekleştirilmesinin düşünülmesini gerektirmektedir. FPGA üzerindeki işlerin çeşitliliği arttıkça bu işlemlerin kontrolü için de yazılım modülleri gerekmektedir.

FPGA üzerinde geliştirmeyi yapanların donanım mühendisi olması durumunda mutlaka yazılım ve donanım mühendislerinin birlikte çalışması gerekmektedir.

Proje süresince gereksinimler, algoritmalar ve donanımlar değişebilmektedir. Bu değişiklikler de

yazılım önemli değişikliklere yol açabilmektedir. Sİ yazılımlarının test edilmesi gömülü yazılımlar olmaları nedeniyle daha çok işgücü gerektirmektedir. Donanımlar olmaksızın yazılımın geliştirilebilmesi için simülatörlerin geliştirilmesi gerekmektedir.



Şekil 3: Sistem tasarım ve geliştirme döngüsü

2.3. Sinyal İşleme Yazılımları Genel Özellikleri

Sİ yazılımlarının karmaşıklığı, projelerdeki geliştirme süresinin kısalığı ile maliyet, performans ve güç kriterlerine göre yapılan ödünleşime göre artmaktadır. Mevcut projelerin değerlendirilmesi sonucu oluşturulan Sİ yazılımlarının genel özellikleri aşağıdaki gibidir.

1. Sİ yazılımlarında donanım ayarlarının yapıldığı sürücü fonksiyonları kullanılmaktadır. Bu fonksiyonların gerçek zaman gerekleri sıkıdır.
2. Sİ yazılımları kontrol işlemci yazılımları ile standart bir arayüzden haberleşmektedir. Bu arayüzlerdeki gerçek zaman gerekleri daha gevşektir.
3. Yazılımlar sinyal işleme fonksiyonları barındırmaktadır. Bu sinyal işleme fonksiyonları FIR, FFT, kodlama, eşikleme, frekans düşürme, vektör ve matris işlemleri ve hedef takip, hedef ayrıştırma, yayın ayrıştırma, yayın izleme, yön, konum bulma, planlama gibi algoritmalardır.
4. Performans gerekleri doğrultusunda birden fazla işlemci üzerinde paralel çalışma gereksinimleri ortaya çıkabilmektedir.
5. Performans gerekleri doğrultusunda ASIC, FPGA veya hazır yonga kullanımı zorunluluğu oluşabilmektedir.
6. Performans gerekleri doğrultusunda işlemcilerin özel veri işleme yeteneklerini ve kütüphaneleri kullanarak optimize kod yazma ihtiyaçları bulunmaktadır.
7. Cihaz İçi Test (CİT) ve Kalibrasyon işlevlerini yerine getiren fonksiyonlar bulunmaktadır.
8. Tüm sinyal işleme fonksiyonlarını, taktik kayıt, CİT ve kalibrasyon işlevlerini kontrol edecek kontrol yazılım bloğu bulunmaktadır.
9. Yazılım güncellemelerini ve yeni kalibrasyon tablolarını yükleme yetenekleri bulunmaktadır.
10. İhtiyaçlar doğrultusunda yüksek hızlı verileri kayıt edebilme yeteneği bulunmaktadır.
11. DSP'de gerçekleşen işlemlerin ve akışın takip edilebilmesi amacıyla, günlük tutma yeteneği bulunmaktadır.

2.4. Hedeflenen Gerekler

2.3 bölümünde belirtilen genel özelliklerin yanında 2.2'de anlatılan geliştirme sürecini rahatlatmak amacıyla mimarinin aşağıdaki ek özellikleri de sağlaması hedeflenmiştir.

1. Konfigüre edilebilir yapıda tasarım,
2. DSP'de gerçekleşen işlemleri ve akışı takip etme amaçlı monitör fonksiyonlarının bulunması ve bu fonksiyonlara dışarıdan erişim,
3. Sİ yazılımlarının simülasyon verileri ile çalışabilmesi için tasarım yapılması ve simülasyon verilerinin Sİ yazılımları tarafından üretilmesi hedeflenmiştir.

Ayrıca aşağıdaki kalite faktörleri hedeflenmiştir.

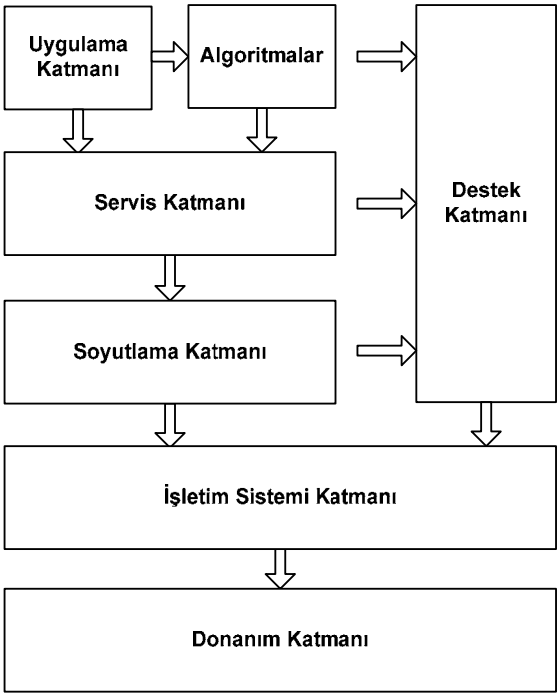
- Kendini Toparlayabilme (Recoverability)
- Çözümlenebilirlik (Analyzability)
- Değiştirilebilirlik (Changeability)
- Kararlılık (Stability)
- Test Edilebilirlik (Testability)
- Uyumlanabilme (Adaptability)
- Kurulum (Installability)

3. Tasarlanan Mimari

Tasarlanan mimari için tek görünüm hazırlanmıştır. Hazırlanan görünüm Krutchen'in 4+1 görünümünde yer alan geliştirme görünümüdür[2]. Mimari katmanlı bir yapıdadır[3]. Hazırlanan geliştirme görünümü yazılım katmanları ve modülleri arasındaki ilişkileri

göstermektedir. Katmanların erişebildiği diğer katmanlar oklarla gösterilmiştir.

Mimaride İşletim Sistemi, Soyutlama, Servis, Uygulama ve Algoritma katmanları ve tüm katmanlara hizmet verebilen Destek katmanı bulunmaktadır. Ayrıca sistemdeki işlemci, veriyolu benzeri tüm donanım bileşenlerini temsil eden ve herhangi bir yazılım barındırmayan bir Donanım katmanı tanımlanmıştır. Örneğin işlemci ve VME, PCI, RapidIO gibi veriyolu donanımları bu katmanda yer almaktadır.



Şekil 4: Sİ Yazılımları Ortak Mimarisi

Mimari oluşturma çalışmaları kapsamında her bir katman içerisinde yer alan modüller de belirlenmiş ve detaylandırılmıştır. Proje gereksinimleri doğrultusunda modüllerden ihtiyaç duyulanlar gerçeklenecek, gerektiğinde yeni modüller eklenebilecektir.

Tasarlanan mimaride en alt katmanda yazılımların koştuğu ve kullandığı donanımların yer aldığı Donanım Katmanı bulunmaktadır.

Donanım katmanı üzerinde İşletim Sistemi katmanı yer almaktadır. İşletim sistemi, donanımlara ve işletim sistemine özel sürücüler ve kütüphaneler bu katmanda bulunmaktadır. İşletim sistemi kullanılan yazılımlarda bu katmanda işletim sistemi, kart destek kütüphanesi gibi hazır temin edilen yazılımlar bulunacaktır. Bununla birlikte, Sİ yazılımları herhangi bir işletim sistemi kullanılmadan da geliştirilebilmektedir. Bu durumda, İşletim Sistemi katmanında donanımlarla haberleşmek için geliştirilecek modüller bulunacaktır.

İşletim sisteminin üst katmanında Soyutlama katmanı bulunmaktadır. Bu katman, Sİ yazılımını işletim sistemi ve donanımlardan olabildiğince soyutlamak amacıyla tasarlanmış olan katmandır. Sİ yazılımlarını işletim sistemi ve donanım birimleri gibi projeden projeye veya zaman içerisinde değişebilecek birimlerden soyutlayıp, kendi içerisinde bağımsız bir hale getirmek hedeflenmiştir. Böylece, yazılımların tekrar kullanılabilmesi, kolayca platformlar arası taşınabilmesi amaçlarına ulaşmak için Soyutlama katmanı tasarlanan Referans Mimarinin en önemli ve çekirdek katmanını oluşturmaktadır.

Soyutlama katmanında, işletim sistemi veya donanım kütüphanelerinde sunulan genel amaçlı fonksiyonların üzerini kaplayarak DSP uygulama yazılımlarına uyumlayan ya da özel amaçlarla yazılmış küçük kütüphane fonksiyonlarının kombinasyonlarıyla oluşturulan genel fonksiyonlardan oluşur.

Ayrıca, Soyutlama katmanında donanım birimleri için simülatörler rahatlıkla geliştirilebilecek, bu sayede donanım değişikliklerinin üst katmanlardaki yazılımları etkilememesi sağlandığı gibi projelerde henüz donanımın hazır olmadığı aşamalarda yazılım geliştirilmeye devam edilebilecektir.

Soyutlama katmanının üst katmanında Servis katmanı bulunmaktadır. Servis katmanı bir üstteki Uygulama katmanına haberleşme yönteminden tamamen soyutlanmış olarak ilgili donanım birimlerini kontrol etmek için gerekli arayüzleri, operasyonel ve yönetimsel servisleri sunar. Sunulan servislere örnek olarak cihaz içi test, hata kontrol, kaynak yönetimi ile veri arayüzü sayılabilmektedir.

En üst seviyede Uygulama katmanı yer almaktadır. Uygulama katmanı Sİ yazılımının ana akış işlemlerinin ve senaryolarının yönetiminin sağlandığı ana kontrol katmanıdır. Bunun yanında, Sİ yazılımının başlangıç işlemleri ile hata yönetim kontrolleri ve kalibrasyon kontrolleri de yine bu katmanda uygulanmaktadır.

Sinyal işleme algoritmaları Uygulama katmanı ile aynı seviyede bulunan Algoritmalar katmanı içerisinde bulunmaktadır. Bu katman Uygulama katmanı tarafından kullanılabilmektedir. Fakat Algoritma katmanından Uygulama katmanına erişim engellenmiştir. Böylelikle algoritmaların yazılımının senaryosal ve yönetimsel işlevleri ile iç içe geçmesi önlenmiştir.

Algoritma katmanının donanımdan soyutlanmış görünmesi nedeniyle algoritmaların hız optimizasyonu bir sorun gibi görünebilir. Ancak algoritmaların İşletim sistemi katmanında bulunan donanıma göre optimizasyonu yapılmış sinyal işleme, veri toplama,

matematiksel işlemler ve benzeri amaçlı kütüphaneleri kullanması Destek katmanı aracılığı ile gerçekleştirilebilmektedir.

Destek katmanı tüm katmanların ihtiyaç duyabileceği ortak işlevleri barındırmaktadır. Destek katmanında yer alabilecek ortak işlevler olarak günlük tutma, hata ayıklama, gözetim işlevleri değerlendirilebilmektedir. Ayrıca, yukarıda anlatıldığı gibi donanıma özel kütüphaneler için de Algoritma katmanıyla kütüphaneler arasındaki arayüzü de sağlamaktadır.

Tasarlanmış olan Sİ Yazılımları Referans Mimarisinin uygulanmasıyla projelerdeki gereksinim ve platform değişikliklerinin daha hızlı gerçekleştirilmesi, ortak bileşen ve kütüphanelerin oluşturulması ile kalite gereklerinin daha rahat karşılanması mümkün olacaktır.

4. Mimarinin Değerlendirilmesi

Sİ Yazılımları Referans Mimarisinin hızlı bir şekilde değerlendirilip doğrulanması için mevcut projelerden seçilen örnek senaryolar kullanılmıştır. Bu senaryolar üzerinden mimari test edilerek eksikler ve sorunlar belirlenmiş ve giderilmiştir. Doğrulama çalışmalarından katman ve modül tanımlarının iyileştirilmesi ve detaylandırılmasında faydalanılmıştır.

Hazırlanan Sİ Yazılımları Referans Mimarisinin doğrulanması amacıyla kullanılan senaryolar başlangıç işlemleri, operasyonel çalışma, kalibrasyon yapma, cihaz içi test, yazılım yükleme, veri kayıt, gözlemleme / hata ayıklama, hata yönetimi, simülasyon verisi ile çalışma ve günlük tutma olarak belirlenmiştir. Bu senaryoların ana akışları adım adım belirlenmiş ve her bir adım mimarinin bir katmanında bulunan bir modül ile ilişkilendirilmeye çalışılmıştır. Böylelikle modüllerin senaryolar kapsamında birbirleriyle olan ilişkileri belirlenmiş ve ilişkiler katmanlı mimari kurallarına göre değerlendirilmiştir. Mimari kurallara uymayan modül ilişkileri ile karşılaşıldığında yeni modüllerin tanımlanması, modül katmanının değiştirilmesi gibi yaklaşımlarla uygunsuz durumlar giderilmiştir. Bir senaryo adımı herhangi bir modül ile karşılanamadığı durumlarda da yeni modüller tanımlanmıştır. Ayrıca modüllerin tanımları, ilişkili oldukları senaryo adımlarının tanımlarından faydalanılarak geliştirilmiştir. Bu yöntemle modüller arası ilişkiler hızlı bir şekilde test edilmiş ve modüllerin daha iyi tanımlandırılması ve doğru seviyelendirilmesi sağlanmıştır.

Sİ Yazılımları Referans Mimarisi çalışmaları çerçevesinde üzerinde durulan ve ortaya çıkartılan katmanlı mimari yapısı ile hedeflenen modülerliğin sağlanması başarılmıştır. Bu şekilde Sİ yazılımlarında karşılaşılan iç içe geçmiş ve ayrıştırılamayan yapıdaki yazılımların dezavantajından kurtulmaya yönelik olarak iyileştirmeler olacağı değerlendirilmektedir.

2.2 bölümünde belirtildiği gibi, projelerin ön tasarım aşamalarında, algoritmaların yazılım ve donanım birimleri arasında paylaştırılması ile algoritmalar için değerlendirilen yazılım bloklarının netleştirilmesi üzerinde çalışmalar yapılmaktadır. Tasarlanan mimarinin, algoritmaların ayrıştırılması ve paylaştırılması ile ortaya çıkan resmin daha üst bir bakış açısıyla yorumlanması açısından da katkı sağlayacağı değerlendirilmektedir.

Her yeni başlayan projede algoritma işlerinin paylaştırılması ile birlikte yazılım-yazılım, yazılım-donanım arayüzleri ve Sİ yazılımın akışı ve işlevlerinin değerlendirilmesiyle ortaya çıkartılan yazılım tasarımları planlanmaktadır. Ortaya konan Sİ Yazılımları Referans Mimarisi ile bu aşamanın hızlıca geçilip proje gereklerinin Mimari üzerine oturtulması, proje zamanlamasını ve maliyetlerini azaltacaktır.

Yine her yeni projenin başlangıcında, proje gereklerinin Sİ Yazılımları Referans Mimari üzerine oturtulmasıyla birlikte bunun doğal sonucu olarak projeler arasında ortaklamaların doğacağı da aşikârdır. Bununla beraber, ortak mimari tasarımla yürüyen projelerde, özellikle Soyutlama, Servis ve Destek katmanlarında ortak modüllerin kullanılması değerlendirilecektir. Tasarımın ortak kullanımı olarak tetiklenen projeler arası ortaklıklarda, hazırlanan kodların tekrar kullanımına gidilebilecektir.

Referans Mimari tanımlamanın diğer bir avantajı da proje ekibine yeni katılan ya da üniversiteden yeni mezun olmuş mühendislerin daha hızlı ve doğru bir şekilde yazılım geliştirme çalışmalarına başlaması olacaktır.

5. Sonuç

Bu bildiride, ASELSAN MST Grubu bünyesinde geliştirilen Sİ Yazılımları Referans Mimarisi için gerçekleştirilen çalışmalar aktarılmıştır. Mimariyi oluşturmak için mevcut ve geçmiş projelerdeki Sİ yazılımları incelenerek ortak özellikler ve gereksinimler belirlenmiştir. Bu gerekler doğrultusunda katmanlı bir yapı temel alınarak Mimari oluşturulmuş ve çeşitli senaryolar ile doğrulanmıştır.

Sunulmuş olan Mimarinin katmanlı yapı tabanlı olması modülerliğin ve soyutlamanın sağlanması açısından faydalı olmuştur.

Sİ yazılımlarının tasarım aşamasına yön vererek hızlandırması da tasarlanan mimariden elde edilen kazanımlar arasında ön sıralara çıkmıştır.

Ortaya konan tasarımın ve gerçeklenecek modüllerin tekrar kullanımı neticesinde proje geliştirme aşamasında zamandan ve işgücünden tasarruf edilerek proje maliyetlerinin azaltılması mümkün kılınmıştır.

Ayrıca, Sİ Yazılımları Referans Mimarisi, ASELSAN'ın bilgi birikimini korumasına, kayıt altına almasına ve yeni katılan personeline kolayca aktarabilmesine katkı sağlamaktadır.

6. Referanslar

[1] TECHNICAL REPORT ISO/IEC 9126-1 Software Engineering - Product Quality

[2] Kruchten, P., "The 4+1 View Model of Architecture," IEEE Software 12(6), 42-50,1995

[3] Buschmann, F., Meunier, R., Rohnert, H., Sornmerlad, P., Stal, M. Pattern Oriented Software Architecture Volume–1, Wiley, 2001, Chichester

ELEKTRONİK HARP SİSTEMLERİ GÖMÜLÜ KONTROL YAZILIMI MİMARİSİ

H.Özgür GÖREN
ASELSAN A.Ş.
hogoren@mst.aselsan.com.tr

Eda GÜRLER
ASELSAN A.Ş.
tverdi@mst.aselsan.com.tr

Öz

Elektronik Harp Sistemleri, grafiksel kullanıcı arayüzünü sağlayan konsol yazılımları, RF sinyalleri toplamaya ve işlemeye yönelik anten ve almaç birimleri, sinyal işleme yazılımları gibi pek çok yazılım ve donanım biriminin bir arada var olduğu karmaşık sistemlerdir. Gömülü kontrol yazılımları tipik olarak sistemde yer alan birimlerin, işleyiş senaryolarını gerçekleştirmek üzere, uyumlu çalışmalarından sorumlu yazılımlar olarak sistemde merkezi bir rol üstlenir. Kontrol yazılımlarının arayüz gerekleri sistem birimlerinin çeşitliliği ile doğru orantılı olarak artmaktadır. Sistemdeki merkezi rolleri gereği kontrol yazılımları sistem gereklerindeki değişikliklere duyarlıdır. Gömülü kontrol yazılımları arayüz ve gereklerdeki değişime açık olmak, çevre birimleri olmaksızın test edilebilmek gibi kısıtlarla karşı karşıyadır. Elektronik Harp Sistemleri Gömülü Kontrol Yazılımları Mimarisinin tasarlanmasında bu kısıtlar dikkate alınırken, yeniden kullanılırlığın arttırılması hedeflenmiştir. Bildiride elektronik harp sistemlerindeki gömülü kontrol yazılımları için tasarlanan mimari anlatılmaktadır.

1 Giriş

Gömülü kontrol yazılımları, bir sistemde yer alan donanım birimlerini kontrol eden, sistemde yer alan diğer gömülü yazılımların grafiksel kullanıcı arayüzü gibi uygulama yazılımları ile arasındaki eşlemeyi sağlayan yazılımlardır. ASELSAN Mikrodalga ve Sistem Teknolojileri (MST) Grubu'nda Radar, Elektronik Harp ve Silah Sistemleri projelerinde kullanılmak üzere bir gömülü kontrol yazılımları mimarisi tasarlanmıştır. Mimari tasarlanırken Radar ve Elektronik Harp projeleri incelenerek mimariyi yönlendiren gereksinimler ortaya konmuştur. Gereksinimleri karşılamaya uygunluğu nedeniyle katmanlı mimari tasarım kalıbı kullanılmıştır[1]. Katmanlar, bileşenlerin tanımlı ve kısıtlı sorumluluklara sahip olmasına imkan verecek şekilde seçilmiş ve yeniden kullanılırlığı arttırması hedeflenmiştir.

Kontrol yazılımlarının Grafik Kullanıcı Arayüzü, REFORM bileşenleri gibi uygulama yazılımları ile aynı dağıtık yapı içerisinde yer alabilmesi Common Object Request Broker Architecture (CORBA) kullanılması ile mümkündür.[4,5] Mimari tasarım CORBA kullanımına uygun şekilde yapılmıştır.

2 Mimariyi Yönlendiren Gereksinimler

Mimari, aşağıdaki gereksinimler dikkate alınarak tasarlanmıştır.

2.1 Değişikliğe Açıklık

Gömülü kontrol yazılımları, pek çok sistem birimimin, sistem işleyiş senaryolarını gerçekleştirmek üzere uyumlu çalışmalarından sorumlu yazılımlar olarak, sistem birimlerine bağımlı yazılımlardır. Tipik olarak kontrol yazılımları çok sayıda arayüzü gerçekleştirmek zorundadır. Farklı projelerde kullanılan farklı yazılım ve donanım birimleriyle ilgili arayüzlere kolay uyumlandırılabilir olmak kontrol yazılımları için temel problemlerden biridir. Ayrıca proje sürecinde sistem birimlerinde, birimlerin fiziksel arayüzlerinde ve kullanılan protokollerde değişikler olabilir. Kontrol yazılımları sistemdeki değişikliklerin yazılıma en az etkiyle hızlı ve doğru bir biçimde yansıtılmasına imkan verecek esnekliğe sahip olmalıdır.

Bunların yanı sıra farklı projelerde benzer birimlerle farklı işlevler gerçekleştiriliyor, ya da proje sürecinde işlevler değiştiriliyor olabilir. kontrol yazılımları işlevsel farklılık ve değişiklikleri uygulamada da gerekli esnekliğe sahip olmalıdır.

2.2 Yeniden Kullanılabilirlik

Kontrol işlemci yazılımları, sistem birimlerine ve sistem işleyiş senaryolarına bağımlı yazılımlar olmakla beraber, ortak yazılım bloklarından oluşurlar. Sistemlerde var olan ortak işlevler, donanım ve yazılım birimleri ve alt seviye arayüzleri bu tür ortak yazılım bloklarının kaynağıdır. Aynı işlevlerin tekrar tekrar kodlanması yerine, yeniden kullanımı, harcanan işgücünü azaltacağı gibi yazılımların güvenilirliğini de arttıracaktır. Bu nedenlerle mimari tasarımın hedeflerinden biri gerekli soyutlamalar yapılarak, yazılım bloklarının yeniden kullanılabilirliği arttıracak şekilde oluşturulmasıdır.

2.3 Test Gerekleri

Yazılımların kalitesini arttırmak amacıyla yazılımları her seviyede test etmek hedeflerimizden biridir. Yazılımları oluşturan alt blokları test etmek, yazılım geliştirme sürecimizin önemli bir parçasıdır. Ancak gömülü yazılımlarda çevre birimler olmadan alt blokları test etmek çoğu zaman zorlu bir süreçtir. Bu nedenle yazılımı oluşturan alt blokları, bağlaşımları (coupling) en az olacak şekilde tasarlayarak blokları tek bir arayüz üzerinden test edilebilir hale getirmek mimari hedeflerinden biri olacaktır.

2.4 Performans

Kontrol yazılımları sistem işleyişini kontrol ettikleri için işleyiş performansları doğrudan sistem performansını etkiler. Kontrol yazılımları sistemdeki hızlı veri yollarına doğrudan bağlı işlemciler üzerinde koşar ve sistemdeki veri akışının önemli bir bölümü bu yazılımlar tarafından kontrol edilir. Hızlı veri akışını sağlarken senaryo yönetimi için gerekli fonksiyonları koşturmak ve çok sayıdaki kontrol arayüzünün gereklerini sağlamak kontrol yazılımlarının gerçek zamanlı çalışmasını zorunlu kılmaktadır. Gerçek zamanlı çalışma, gerçek zamanlı girdilere sınırlı zamanda tepki verebilmek anlamına gelir. Bu nedenle kontrol yazılımları çok işlevli bir şekilde tasarlanmalı ve tasarım, sistemdeki bağımsız işlevlerin birbirini beklemesine neden olmamalıdır. İşletim sistemi üzerinde paralel çalışan kollara(task) dönüşen aktif nesneler çok işlevli şekilde çalışmayı sağlarken, sisteme gereksiz yük getirmeyecek şekilde seçilmelidir.

2.5 Hata Yönetimi

Hata yönetimi, basit haliyle sistemde oluşan hatanın algılanması ve giderilmesini içerir. Kontrol yazılımları sistem birimleri arası eşleme ve arayüzleri sağlayan yazılımlar olması itibariyle hataya açık bir yapıdadır. Çevre birimleri, eşleme bozulmaları ve kontrol yazılımı kaynaklı hatalar etkili biçimde toplanarak, mümkün olan yerlerde hata giderici eylemler koşturulabilmelidir. Bu nedenlerle mimari hata yönetimini kolaylaştıracak şekilde tasarlanmalıdır.

2.6 Rapor Tutma

Sistem çalışmasının raporlanması performans ölçümü ve olası hata kaynaklarının bulunması açılarından önemlidir. Donanım birimleri ve gömülü yazılımların rapor tutma olanağı oldukça kısıtlıdır. Kontrol yazılımları donanım birimleri ve gömülü yazılımlarla doğrudan arayüz sağlayan yazılımlar olarak raporlamayı gerçekleştirecek yapıda tasarlanmalıdır.

3 MİMARİ

3.1 Katmanlı Mimari

Katmanlı mimari, tasarım sonucu oluşan paketleri soyutlama seviyelerine göre belli bir hiyerarşik yapıda organize etmektedir[1]. Katmanlı mimarinin seçilmesi, yazılım bloklarını farklı katmanlarda toplayıp soyutlama sağlamakta ve yeniden kullanımı kolaylaştırmaktadır.

3.2 Katmanlar

Gömülü Kontrol Yazılımları için dört katmanlı bir mimari belirlenmiştir. Bu katmanlar Protokol, Sistem Soyutlama, İşlev ve Sistem Denetimi katmanlarıdır.

Protokol Katmanı: Sistemdeki birimlerle haberleşmekte kullanılan arayüzleri gerçekleyen sınıfların bulunduğu katmandır. Seri Kanal, UDP Soket gibi alt seviye haberleşme arayüzünü sağlayan sınıflar bu katmandadır. Bu katmandaki sınıflar veri yolundan gelen tek ve bütün bir mesajın ham veri olarak sistem soyutlama seviyesine iletilmesinden sorumludur. Diğer sorumluluğu da sistem soyutlama

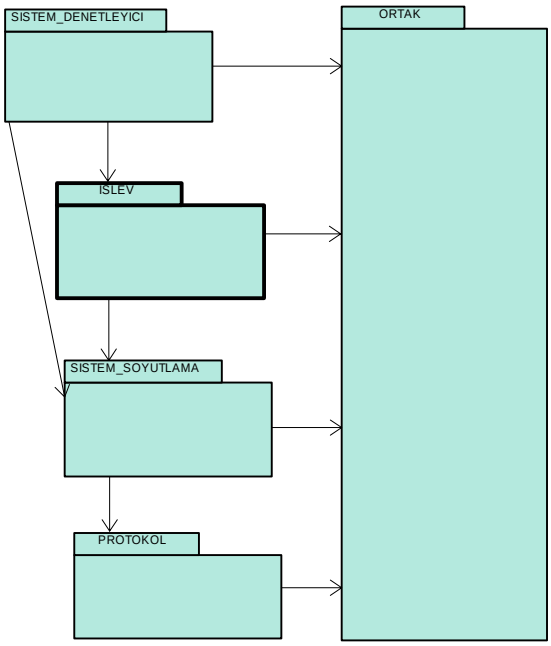
seviyesinden gelen ham verinin tek ve bütün olarak veri yolundan gönderilmesidir

Sistem Soyutlama Katmanı: Kontrol yazılımının haberleştiği birimlerle arasında giden gelen mesajların işlendiği seviyedir. Çevre yazılım ve donanımlardan gelen mesajlar bu seviyede ayrıştırılır. Her bir sistem birimi için bir paket bulunmaktadır. Ayrıca sistem birimlerine özgü haberleşme protokolleri bu seviyede sınıflarla karşılanır. Bir sorumluluğu Protokol seviyesinden gelen ham mesajın kontrol yazılımı için anlamlı bir yapıya dönüştürülmesidir. Diğer sorumluluğu ise işlev seviyesinden gelen isteklerin ham mesaja dönüştürülüp protokol seviyesine gönderilmesidir.

İşlev Katmanı: Kontrol yazılımının işlevsel sorumluluklarının yerine getirildiği bir katmandır. Kontrol yazılımın bir senaryo dahilinde yönettiği işlevler bu katmanda yer alan aktif sınıflarca yönetilir.

Sistem Denetleyici:. Sistem denetleyici, sistem soyutlama katmanında birbirinde soyutlanmış olan birimler arasında ve bu birimlerin işlev sınıfları ile arasındaki iletişimi yönlendirir. Hata raporlama gibi sistem birimleri arasındaki ortak işlemleri gerçeklemek için kullanılır.

Yazılım mimarisinde her katman ayrı paketler içerinde yer alır. Bunlara ek olarak çeşitli seviyelerde kullanılan ortak tip ve tanımların yer aldığı Ortak Paket bulunmaktadır.



Şekil 1 Katmanlar Arası İlişkiler

3.3 Mimari Kurallar

Mimari tasarımda katmanlar arası ve katman içi paketler arası ilişkileri tanımlayan kurallar belirlenmiştir. Bu kurallar şu şekilde sıralanabilir:

1. Alt katmanlar üst katmanlara bağımlı olamaz. Alttan üste doğru katmanlar şu şekilde sıralanmıştır: Protokol, Sistem Soyutlama Seviyesi, İşlev ve Sistem Denetleyici.
2. Sistem Denetleyici dışındaki katmanlar yalnızca bir alt katmana ulaşabilir.
3. Protokol ve Sistem Soyutlama seviyesindeki paketler bağımsız arayüz ve haberleşme protokollerini tanımladıklarından birbirlerine bağımlı olmamalıdır.
4. Bir protokolu ya da sistem birimini temsil eden bütün paketlerin soyut arayüzü olacaktır. Bu sınıf paketin dışarı verdiği servisleri temsil etmektedir.
5. Tüm seviyelerdeki paketler ortak pakete bağımlı olabilir.

4 Katman Yapıları

4.1 Protokol

Kontrol yazılımı ile diğer sistem birimleri arasındaki haberleşmeyi sağlayan alt seviye arayüzler bu katmanda yer almaktadır. Her alt seviye arayüz için Protokol katmanında ayrı bir paket bulunmaktadır. Bu katmanda yer alan paketler diğer katmanlara veya birbirlerine bağımlı değildirler. Protokol katmanında, bu katmandaki sınıfların gerçeklemesi gereken arayüzü ve bir katmanla ilişkileri tanımlayan Haberleşme Yapısı Paketi bulunmaktadır. Haberleşme Yapısı, katmanda yer alan sınıfların sunduğu arayüzü tanımlayan IHaberlesme ve IHaberlesme sınıfı tarafından gönderilen mesajı çözmek ile sorumlu olan IKullanici sınıflarından oluşur. Sistem Soyutlama Katmanındaki birimler kullandıkları haberleşme sınıfı için Kullanıcı arayüzü gerçeklerler. Seri Kanal, UDP, TCP gibi haberleşme sınıfları IHaberlesme sınıfından türetilir.

Protokol katmanının tasarımı şu özellikleri sağlamıştır:

- Haberleşme sınıfları birbirinden bağımsız paketler içerisinde yer alır, bağımsız olarak projeye eklenip çıkarılabilir.
- Bir sistem biriminin haberleşme arayüzü değişse bile(örn. TCP'den UDP'ye) değişiklik yaratılma anında karşılanacaktır.
- Haberleşme sınıflarının arayüzden bağımsızlığı sayesinde gerçek haberleşme ortamı olmadan simülasyon sınıfları kullanmak mümkün olmaktadır. (örn. UDP'den yollamak yerine dosyaya yazmak, ya da arayüzü konsola yönlendirmek)

4.2 Sistem Soyutlama

Kontrol yazılımının haberleştiği her sistem birimine karşılık bu katmanda bir paket yer alır. Sistem birimine özgü arayüz gerekleri bu paketlerde gerçeklenir. Sistem birimleri Yazılım ve Donanım olarak iki ayrı pakette toplanmıştır. Her birim Protokol katmanından ilgili arayüzle ilişkisi kurmak üzere Haberleşme yapısındaki Kullanıcı rolünü gerçekler.

4.2.1 Donanım

Donanım sınıfları diğer sistem birimleri gibi Haberleşme Yapısını gerçeklemelidir. Bunun yanı sıra ASELSAN'da tasarlanan birimler ortak bir haberleşme protokolüne sahiptir. Bu Protokole sahip birimler Donanım Haberleşme Yapısında yer alan arayüzleri gerçeklerler. Donanım Haberleşme protokolü Donanım Haberleşme sınıfı tarafından gerçeklenir.

4.2.2 Yazılım

Kontrol yazılımının haberleştiği her yazılıma karşılık bir paket yer alır. Yazılıma yollanan komutlar yazılım arayüz sınıflarında soyutlanırken, yazılımdan gelen istekler yazılım kullanıcı sınıfı tarafından Sistem Denetleyiciye iletilir.

Sistem Soyutlama katmanının tasarımı şu özellikleri sağlamıştır:

- Protokol katmanında yalnızca Haberleşme Yapısına bağımlı, diğer paketlerden bağımsızdır.
- Sistem birimleri birbirinden bağımsız paketler içerisinde yer alır, bağımsız olarak projeye eklenip çıkarılabilir.
- Geliştirme safhasında sıklıkla karşılaşabileceği gibi tüm sistem birimleri bir arada olmadan da, olmayan birimler için simülasyon sınıfları kullanılarak, yazılım-yazılım ve yazılım-donanım entegrasyonu yapılabilir.
- Sistem soyutlama seviyesindeki sınıflar sistem işleyiş senaryolarından çok sistem birimlerine yönelik tasarlandığından bir ölçüde işlevlerden bağımsızdırlar. İşlevlerdeki değişiklikler sistem soyutlama seviyesini etkilemeyeceği gibi bu seviyedeki paketlerin sistem birimlerinin kullanıldığı farklı projelerde yeniden kullanılma imkanı arttırılmıştır.

4.3 İşlev Katmanı

Kontrol yazılımının bir senaryo dahilinde yönettiği işlevler bu katmanda yer alan sınıflarca yönetilir. Sistem birimleri arası eşlemeyi yönetmek İşlev katmanındaki sınıfların temel görevlerinden biridir. Bu açıdan İşlev katmanında yer alan sınıflar sistem senaryolarına doğrudan bağımlıdır. Sistemdeki işlevler, birbirini etkileyen yapıda olabileceğinden, diğer katmanlardan farklı olarak bu katmanda yer alan paketler ve sınıflar arası ilişki kurulabilir. Sistem Denetleyici seviyesini bu katmandaki işlevler ve bunların arası ilişkilerden bağımsız kılmak üzere Facade tasarım kalıbına[2] benzer şekilde İşlev Fasat adı verilen bir sınıf kullanılır. Bu sınıf işlev sınıflarının arayüzlerini üst katmandan gizler.

İşlev katmanının tasarımı şu özellikleri

sağlamıştır:

- İşlevler sistem birimlerinin gerçek implementasyonlarından bağımsızdır.
- İşlevler sistem senaryoları gerektirmedikçe birbirlerinden bağımsızdır.
- Sistem senaryolarında değişiklikler büyük oranda bu katmanda yer alan sınıflarca karşılanır.

4.4 Sistem Denetleyici

Sistem Denetleyici, en üstte yer alan katmanı ve bu katmanda yer alan Sistem Denetleyici sınıfını betimler. Sistem denetleyici sınıfı, sistem soyutlama katmanında birbirinde soyutlanmış olan birimler arasında ve bu birimlerin işlev sınıfları ile arasındaki iletişimi yönlendirir. Sistem Denetleyici sınıfı tüm sistem birimlerinin kullanıcısı olarak bu birimlerden gelen istekleri toplar ve mimari içerisinde katmanlara uygun şekilde dağıtır. Sistem birimlerinin ilettikleri hata raporlama, sürüm bildirme gibi istekler bu sınıfta ortaklanır.

Sistem Denetleyici tasarımı şu özellikleri sağlamıştır:

- Sistem Soyutlama Seviyesinde yer alan sınıflar birbirinden bağımsızken, olası iletişim Sistem Denetleyici üzerinden sağlanmaktadır.
- Kullanıcı sınıflarındaki ortak methodlar tek bir sınıfta gerçeklenmektedir.

5 Mimaride CORBA'nın Yeri

Elektronik Harp projelerinin boyutları ve karmaşıklıkları gereği pek çok heterojen ortamın birlikte çalışmasına ihtiyaç duyulmaktadır. Bu ihtiyaç dağıtık uygulama teknolojilerinin kullanılmasını zorunlu kılmaktadır. Kontrol yazılımlarının diğer uygulama yazılımları ile aynı dağıtık yapı içerisinde yer alabilmesi CORBA'nın kullanılması ile mümkündür.

CORBA kullanılan projelerde, CORBA objelerinin sunduğu operasyonların ve bunların parametrelerinin isimleri, bir Interface Definiton Language (IDL) dosyası haline getirilir. Bu IDL dosyası, sunucu olan CORBA objesinden operasyon çağıracak olan istemci tarafından kullanılır. Kontrol yazılımı mimarisinde IDL dosyalarının oluşturulduğu paket, mimari katmanların dışında tutulmuştur. Bu paket başka hiçbir pakete bağımlı değildir. Ancak, kontrol yazılımının sunucu rolünde olduğu objeler bu IDL oluşturan sınıflarla kalıtım ilişkisi içindedirler ve sundukları operasyonların içini doldururlar. Benzer şekilde, kontrol yazılımının istemci rolünde olduğu objelerin operasyonları da bu IDL sınıfları üzerinden çağırılır.

Sistem Soyutlama katmanında, kontrol yazılımının istemci rolünde olduğu objelerin soyut arayüz sınıfları bulunmaktadır. Diğer uygulama yazılımlarından bir method çağrılacağı zaman bu arayüz sınıflarının operasyonları kullanılır. Böylece, kontrol yazılımının geneli için kararlaştırılmış olan "Alt katmanlar, üst katmanlara bağımlı olamaz" tasarım ilkesi bozulmamış olur. Kontrol yazılımı koşarken, genelde İşlev ya da Sistem Denetleyici katmanındaki paketlerin sınıflarının, uygulama yazılımların operasyonlarını çağırmaları gerekir. Bu durumda, kendilerinden bir alt katman olan Sistem Soyutlama katmanındaki soyut arayüz sınıflarından istedikleri operasyonu çağırırlar.

Yukarıda bahsi geçen soyut arayüz sınıflarıyla kalıtım ilişkisi içinde olan ve IDL sınıfları üzerinden, sunucunun operasyonlarının çağırılmasını sağlayan sınıflar vardır. Bu şekilde, gerek sunucu gerekse istemci olarak Kontrol yazılımı ile CORBA arayüzü olan her yazılım için bir sınıf ayrılmıştır. Bu sınıflarda, Object Request Broker (ORB) ile ilgili başlatma işlemleri, İsimlendirme Servisi kullanılıyor ise bu servisin bulunması, gereken durumlarda bu servise istemcinin bildiği isimle kayıt olunması, gereken durumlarda bu servis üzerinden belirli bir isimle kayıtlı olan sunucunun bulunması gibi işler de gerçekleştirilmiştir.

6 Sonuç

Gömülü kontrol yazılımı mimarisi, elektronik harp projelerinde hayata geçirilerek tipik yazılım gerekleriyle mimarinin uyumu denenmiştir. Haberleşme ortamları Protokol katmanında, sistem birimleri Sistem Soyutlama Seviyesinde, işlem akış senaryoları İşlev seviyesindeki sınıflarca başarıyla karşılanmış, gerekli ortaklamaların yapılabildiği, gereklerdeki değişikliklerin yalnızca ilgili seviyeyi etkilediği, ayrı paketlerde tanımlı arayüzler üzerinden bağımsız geliştirme yapılabildiği, her seviyede simülasyon ve test yapılabildiği görülmüştür.

Mimari tasarıma bakıldığında alt seviyede yer alan Protokol ve Sistem Soyutlama katmanlarında yer alan bileşenlerin az sayıda bağımlılığı olduğu görülmektedir. Üst katmanlara doğru bileşenlerin bağımlılık sayısı artmaktadır. Bu durum, alt seviyedeki bileşenlerin yeniden kullanılırlığı yüksekken, İşlev ve Sistem Denetleyici katmanındaki bileşenlerin yeniden kullanılırlığının görece az olduğunun bir göstergesidir.

Mimaride "Mediator" tasarım kalıbına[2] benzer bir rolü olan Sistem Denetleyicinin boyutu, sistem birimlerinin sayısındaki artışla doğru orantılı olarak artmaktadır. Bu açıdan Sistem Denetleyicinde yer alan methodların basit tutulması önem kazanmaktadır. Mimari, Sistem Denetleyicinin merkezi rolünün hafifletilmesi ve arayüzlerde Port yapısının kullanılması açılarından geliştirilebilir.

7 Referanslar

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sornmerlad, M. Stal of Siemens AG, Germany., Pattern Oriented Software Architecture Volume–1,

[2] E. Gamma, ,R. Helm, R. Johnson, J. Vlisslides "Elements of Reusable Object-Oriented Software", Erich Gamma, Addison-Wesley, 1998,

[3] R.C. Martin, "Design Principles and Design Patterns", www.objectmentor.com

[4] ASELSAN MST Grubu, REFoRM – RADAR Elektronik Harp Fonksiyonel Referans Modeli, 2004.

[5] Object Management Group, CORBA Specification Version 3.0.3, March 2004
http://www.omg.org/cgi-bin/doc?formal/04-03-01

Bileşen Yönelimli Yazılım Tasarımı

EKLENTİ MİMARİSİ VE YAZILIMLARDA EKLENTİ YÖNETİMİ

Ümit Demir
ASELSAN A.Ş.
udemir@mst.aselsan.com.tr

Koray Kadıoğlu
ASELSAN A.Ş.
kadioglu@mst.aselsan.com.tr

Öz

Bir yazılımın yeni sürümü hazırlandığında kaçınılmaz olarak kullanıcılar zamanla daha fazla yetenek isteyeceklerdir. Buna bağlı olarak da geliştiriciler bu istekleri karşılamak üzere yazılıma yeni yetenekler ekleyeceklerdir. Bu bildiri, genişleyebilir bir yazılım mimarisi olan eklenti mimarisi ve uygulama yazılımının genişletilmesine olanak sağlayacak bir adaptör bileşeni tasarımı ve gerçeklemesi sunmaktadır.

1. Giriş

ASELSAN Mikrodalga ve Sistem Teknolojileri (MST) Grubu içinde geliştirilen Uygulama Yazılımı Genişleme Adaptörü (UYGAR) bileşeni yardımıyla yazılımda genişlemeye açık noktalar tanımlanabilecek ve çalışma zamanında bu açık noktaları gerçekleyen eklentiler kullanılabilecektir. Bu kullanım yazılımdaki tanımlı ve açık noktaların doldurulması anlamına gelse de her genişleyebilirliği tanımlamak mümkün olamayacaktır. Bu durumda UYGAR, sadece tanımlı boşlukları dolduran eklentilerin değil, bunun yanısıra tanımlanmamış güncellemeleri gerçekleştiren eklentilerin de kullanılmasını sağlayacaktır.

Önerilen mimari, herhangi bir uygulama yazılımının UYGAR yardımıyla parçalı olarak geliştirilmesini ve lego parçalarının bir araya getirilmesi gibi, bu parçaların kontrollü olarak birleştirilerek yazılımın kolayca hazırlanmasını sağlayacaktır.

Bu bildiride, genişleyebilir bir yazılım mimarisi olan eklenti mimarisi, UYGAR bileşeni tasarımı ve gerçeklenmesi ile eklenti mimarisinin değerlendirilmesi anlatılmaktadır. Örnek kodlar java programlama dilinde [5] verilmiştir.

2. Eklenti Mimarisi

Bir yazılımın yaşam süreci teslim edildikten sonra sona ermemekte, kullanıcılar zamanla yeni isteklerde ve hata bildirimlerinde bulunmaktadırlar. Geliştiriciler de bunları karşılamak üzere hataları gidermekte ve yeni yetenekleri yazılıma eklemektedirler. Geleneksel yöntemler ile bu işlem var olan kodların gözden geçirilerek hataların giderilmesi ya da yazılımda yeni yeteneklerin kodlanması şeklinde olmaktadır. Bu amaçla yazılımın tüm kodları yeniden derlenmekte ve tüm yazılım yeniden test edilmektedir. Bu da zaman kaybına yol açmaktadır. Bu zorluklar ile baş edebilmek amacıyla bileşen yönelimli yazılım geliştirme metodu geliştirilmiştir. Her bileşen belirli bir grup yeteneği içeren kod parçası olup belirli bir arayüzü gerçeklemektedir. Önerilen eklenti mimarisi temelde nesne yönelimli yazılım mimarisidir [3], fakat her bir bileşen seçimliktir.

Dokümanda önerilen mimari, temel olarak Michael Pilone'nin [1]'de belirttiği şekliyle eklenti mimarisi üzerine kurulmuştur. Eklentiler, tek başlarına çalıştırılmaktansa başka bir yazılım tarafından yüklenen, yazılıma yeni yetenekler ve yeni kaynaklar kazandırmak suretiyle yazılımı genişleten yazılım bileşenleridir.

Eklenti mimarisi, uygulamayı geliştiren geliştiricilerin haricinde başka geliştiricilerin de yazılımın kaynak koduna erişmeden ve tüm uygulamayı yeniden derlemeden, yazılıma yeni yetenekler eklemesine ya da var olan yetenekleri değiştirmesine olanak sağlamaktadır. Ayrıca bulunan hataları kolayca yamamak, güvenlik açıklarını kapatmak, yeni servisler eklemek ve var olan servisleri kapatmak ya da güncellemek de mümkün olabilmektedir.

Bundan sonraki alt bölümlerde eklenti mimarisini desteklemek amacıyla geliştirilen UYGAR bileşeninin nasıl gerçekleneceği ve genişleyebilir mimariye sahip bir yazılım tarafından nasıl kullanılacağı anlatılmaktadır.

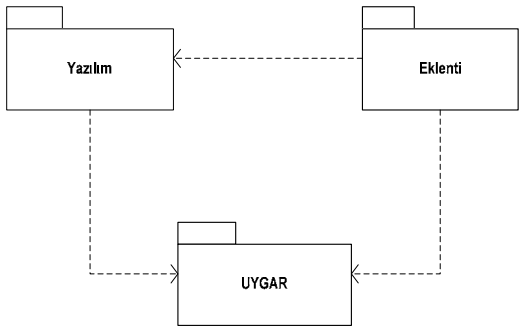
2.1. Uygulama Yazılımı Genişleme Adaptörü

Uygulama Yazılımı Genişleme Adaptörü (UYGAR) bir yazılımın eklentiler ile genişletilebilmesini sağlayan

yazılım bileşenidir. Aşağıdaki alt başlıklarda bu adaptörün nasıl gerçeklenebileceği ve eklenti mimarisindeki yeri anlatılacaktır.

2.1.1. UYGAR Arayüzü Tanımlanması

UYGAR bileşeni çalışma zamanında eklentileri bulacak, yükleyecek, çalıştıracak ve durduracaktır. Şekil 1'de UYGAR bileşeni, bu bileşeni kullanarak genişleyecek yazılım ve eklentiler arası bağımlılıklar gösterilmektedir.



Şekil 1. Bileşenler Arası Bağımlılıklar

Eklenti, uygulama yazılımındaki genişleyebilir noktaların arayüzlerine ve uygulama yazımının eklentilere açtığı verilere ulaşmak için uygulama yazılımına bağımlılık duyar. Ayrıca eklenti arayüz tanımı gerçeklemesi gerektiği için de UYGAR bileşenine bağımlıdır. Uygulama yazılımı da çalışma zamanında eklentileri yönetebilmek için UYGAR bileşenini kullanır.

En temel haliyle bir genişleme adaptörü arayüzü aşağıdaki gibidir:

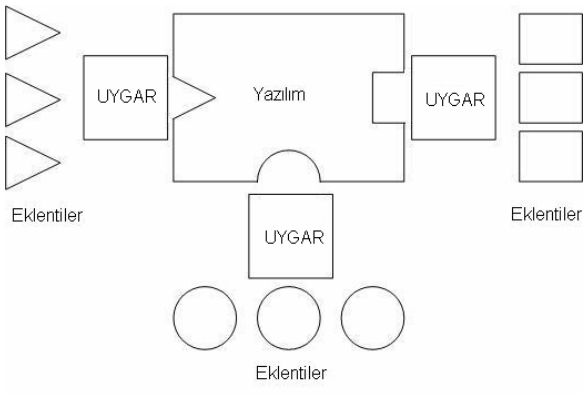
```
public interface GenislemeAdaptoruArayuzu {
   public void ekletileriBaslat();
   public void eklentileriDurdur();
   public void eklentiBaslat(String eklentiSinifAdi);
   public void eklentiDurdur(String eklentiSinifAdi);
   public Object[] ekletileriAl();
   public Object eklentiAl(String eklentiSinifAdi);
   public Graph bagimlilikAgaciAl();
}
```

2.1.2. UYGAR Kullanılarak Eklenti Yüklenmesi

UYGAR, yazılım tarafından üç farklı tipte eklenti yüklenmesinde kullanılabilecektir. Birincisi, yazılımda tanımlı açık noktaların arayüzlerini gerçekleyen eklentilerin kullanılmasını sağlamak, ikincisi hem tanımlı arayüzler hem de yazılımın bazı verilerini kullanan eklentilerin yüklenmesini sağlamak, üçüncüsü de sadece yazılımın bazı verilerini kullanan eklentilerin yüklenmesini sağlamaktır.

2.1.2.1. Sadece Tanımlı Açık Noktalara Uyan Eklentilerin Yüklenmesi

Bu kullanımda UYGAR, yazılımda açık bırakılan arayüzleri gerçekleyen eklentileri yazılıma dahil etmek suretiyle genişleme sağlamaktadır. Şekil 2'de genişleyebilir bir yazılım ve UYGAR bileşenleri ile uygun eklentilerin uygun noktalarda kullanımı gösterilmektedir.



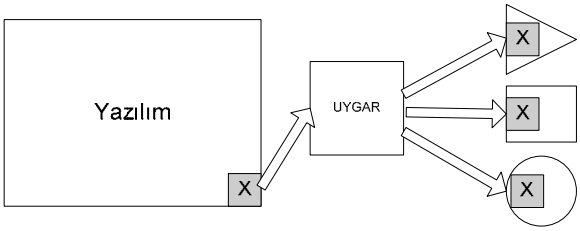
Şekil 2. UYGAR Kullanarak Tanımlı Eklentilerin Yazılımı Genişletmesi

Yazılım, her UYGAR bileşenine, tanımlı bir açık nokta için bekleyeceği arayüzü ve eklentilerin aranacağı dizini tanıtacaktır. UYGAR bileşeni belirtilen dizinden eklenti arayüzüne sahip eklentileri bulacak, çalışma zamanında yükleyecek ve yazılım ile bağlantısını kuracaktır.

Bu kullanımda yazılım, eklentilerin hangi işlevleri gerçekleştirdiğini önceden bilecektir. Hiçbir yazılım verisi eklenti ile paylaşılmadığından, yapılabilecek genişlemeler sadece birbirinden ve çalışma zamanında oluşacak yazılım verilerinden bağımsız olabilecektir.

2.1.2.2. Sadece Yazılım Verilerini Kullanan Eklentilerin Yüklenmesi

Her zaman tüm genişleme noktalarını tanımlamak mümkün değildir. Bu durumda UYGAR tüm eklentileri yükleyecektir. Şekil 3'te genişleyebilir bir yazılım ve UYGAR bileşenleri kullanılarak eklenti kullanımı gösterilmektedir:

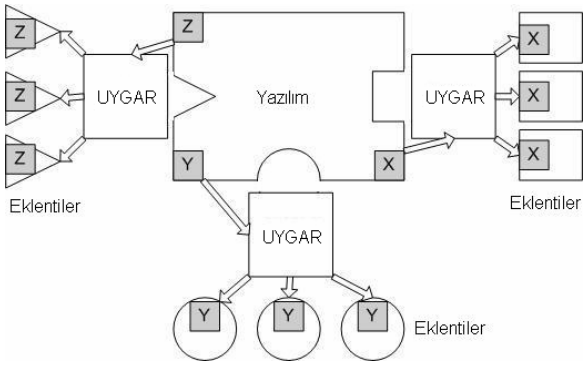


Şekil 3. Sadece Yazılım Verileri Kullanan Eklentilerin Yüklenmesi

Bu kullanımda eklentiler arasında bir gruplama söz konusu değildir. Yazılım, eklentilere kendi iç verilerini UYGAR bileşenleri üzerinden çalışma zamanında aktaracaktır. Bu kullanımda kontrol eklentilerdedir. Eklenti yazılımın kendisine aktardığı veriler üzerinde herhangi bir değişikliği yapabilecektir. Örneğin eklenti bir harita uygulamasında önceden yüklü sembolojiyi iptal edip, yazılımı başka bir semboloji kullanımına geçirebilir ya da semboloji desteğini tamamen ortadan kaldırabilir. Burada önemli olan yazılımdan eklentilere açılacak verileri tanımlamaktır. Tüm verileri açmak eklentiler ile sınırsız güncelleme yapabilme, fakat asgari kontrol anlamına gelecektir. Bu iki parametreyi göz önünde bulundurarak verilerin detayı tanımlanmalıdır.

2.1.2.3. Tanımlı Açık Noktaları ve Yazılım Verilerini Kullanan Eklentilerin Yüklenmesi

Bu kullanımda ise 2.1.2.1 ve 2.1.2.2 başlıklarında verilen kullanımlar birlikte değerlendirilerek her ikisinin de avantajları kullanılmıştır. Bu sayede yazılım, kontrolü kendi içerisinde tanımladığı arayüzleri kullanarak 2.1.2.2'deki durumdan daha fazla elinde tutabilecektir. Ayrıca eklentilerin yazılım verilerine erişimine de kısmi olarak izin vermek yoluyla, onların 2.1.2.1'deki yazılım verilerinden bağımsız olarak güncelleme yapmaları kısıtını da ortadan kaldıracaktır. Şekil 4'te genişleyebilir bir yazılım ve UYGAR bileşenleri ile eklenti kullanımı gösterilmektedir:



Şekil 4. Açık Noktalara Uyan ve Yazılım Verisi Kullanan Eklentilerin Yüklenmesi

2.2. Eklentiler

Eklentiler, yazılıma yeni yetenekler ve yeni kaynaklar kazandırmak suretiyle yazılımı genişleten özel yazılım bileşenleridir. Aşağıdaki alt başlıklarda eklentilerin nasıl gerçekleneceği anlatılmaktadır.

2.2.1. Eklenti Arayüzü Tanımlanması

Eklentilerin, yazılımın çalıştırılması esnasında yüklenmesi ve yazılıma dahil edilmesi gerekmektedir. Yazılım eklentileri UYGAR bileşeni yardımıyla yönetecektir.

Eklenti arayüzü UYGAR bileşeni içinde tanımlanmaktadır. Eklenti arayüzünde temel olarak iki metot tanımlanmış olmalıdır: "baslat" ve "durdur" metotları. UYGAR, eklentiyi çalıştırmak amacıyla "baslat" metodunu çağıracak, eklentiyi durdurmak için de "durdur" metodunu kullanacaktır. "baslat" metodu içerisinde ilkleme, "durdur" metodu içinde de temizleme kodlarını barındırmak eklentinin sorumluluğundadır.

Aşağıda bir eklentinin temel arayüzü tanımlanmaktadır:

```
public interface EklentiArayuzu{
        public void baslat(EklentiVeriTipi
eklentiVeri, GenislemeAdaptoruArayuzu adaptor);
        public void durdur();
}
```

2.2.2. Eklentilerin Paketlenmesi

Eklentilerin, etkin olabilmeleri için, çalışma zamanında yazılıma dahil edilmeleri gerekmektedir. Bu amaçla eklentileri oluşturmak için sınıflar ve kaynak dosyalar paketlenir. Eklenti ayrı bir dosya ya

da dosya grubu olarak paketlenecektir. Paketlemek için çeşitli formatlar kullanılabilir. Örneğin, java programlama dili ile çalışılması durumunda "jar" dosya formatını kullanmak, java altyapısında bu formatı okumak ve işlemek üzere yardımcı sınıflar olması sebebiyle, iyi bir seçim olacaktır.

Her eklentiye ait sınıfların ve kaynakların yanında, eklenti hakkında bilgiler içeren özel dosyalar da paket içine eklenir. Bu özel dosyalar tanım dosyalarıdır. Bu dosyalarda eklentinin geliştirme tarihi, geliştiren kişi, çalıştırılacak sınıf adı, bağımlılıkları gibi, eklenti hakkında temel bilgiler bulunur.

Örnek bir tanım dosyası şöyle olabilir:

```
Manifest-Version: 1.0
Eklenti-Sinifi: AraclarMenuEklentisi
```

Bu örnekte her satır, UYGAR bileşeninin kullanacağı değişken adı ve değeri ikililerinden oluşmaktadır. Örneğin "*Eklenti-Sinifi*" bir değişken adıdır ve UYGAR tarafından kullanım amacı bu eklentinin ana sınıfının ismini almaktır. Bu değişkenin bu eklenti için değeri *AraclarMenuEklentisi'*dir. Tanım dosyasının formatı ve içereceği bilgiler geliştirme esnasında ihtiyaca göre detaylandırılabilir. Bu şekilde UYGAR bileşeni eklentiler hakkında bilgilere ulaşmakta, sadece uygun eklentileri çalıştırarak yazılımın kararlı durumda kalmasını sağlamaktadır. Tanım dosyası olmayan bir eklentinin temel bilgileri tanımsızdır ve UYGAR tarafından göz ardı edilmektedir.

2.2.3. Eklentilerin Yerleştirilmesi

UYGAR, eklentileri çalışma zamanında bulmalıdır. Basit bir yöntem, eklentileri UYGAR bileşenine tanıtılan özel bir dizine kopyalamaktır. UYGAR çalışırken bu dizini tarayıp varsa uygun eklentileri yükleyecektir.

2.2.4. Eklentilerin Gerçeklenmesi

Bölüm 2.1.2'de anlatılan eklenti çeşitlerinin gerçeklenmesi aşağıdaki başlıklarda anlatılmaktadır.

2.2.4.1. Sadece Yazılımda Tanımlı Noktalara Uyan Eklentilerin Gerçeklenmesi

Yazılımdaki genişleme noktaları arayüzler tanımlanarak belirtilmiştir. Örneğin menüye bir ekleme yapmak isteyen eklentiler olabileceği öngörülerek *MenuArayuzu* adında bir arayüz hazırlanabilir.

```
public interface MenuArayuzu{
        public JMenuItem menuElemanıAl();
}
```

Yazılım, menüye ekleme yapacak eklentilerin bu arayüzü gerçeklemiş olmalarını bekleyecektir. Arayüze uygunluk kontrolünü UYGAR bileşeni yapacaktır. Yazılım, eklentinin menuElemaniAl() metodunu kullanarak, çalışma zamanında menüye eklentinin ürettiği menü elemanını ekler. Menüye eleman eklerken tüm kontrol ana yazılımdadır. Eklenti sadece ana yazılımdaki bir boşluğu doldurmuştur. Aşağıdaki örnek menüye eleman eklemek üzere hazırlanmış bir eklenti kodudur:

```
public class AraclarMenuEklentisi implements
EklentiArayuzu, MenuGuncellemeArayuzu{
// Eklenti arayüzünden gelen metodlar
public void baslat(){
        // ilkleme kodları.
}
public void durdur(){
        // temizleme kodları.
}

// yazılım tarafından gerekli menü elemanını almak için
//çağrılan metoddur.
public menuElemanıAl (){
        ...
}
}
```

2.2.4.2. Sadece Yazılım Verileri Kullanan Eklentilerin Gerçeklenmesi

Yazılım, iç verilerini eklentilere açmak suretiyle, eklentilerin çalışma zamanında bu veriler üzerinde güncelleme yapmasına olanak sağlamaktadır.

Örnek bir eklenti kodu aşağıda verilmektedir. Bu kod parçasında yazılım eklentiye kendi iç verilerini geçirmektedir. Eklenti de çalışma zamanında, bu veriler üzerinden yazılımda güncellemeler yapmaktadır. Örnekte eklenti kodu yazılımda tanımlı sembolojiyi yenisi ile değiştirmektedir.

```
public class ASembolojiEklentisi implements
EklentiArayuzu{
public void baslat(Object eklentiVeri){
  // yazılımda değişiklik yapılır.
  sembolojiGuncelle();
}
public void durdur(){
}
public sembolojiGuncelle (){
   //eklenti verisini kullanarak yazılıma eklemeler yapılır.
  ...
}
}
```

2.2.4.3. Tanımlı Arayüzleri Gerçekleyen ve Yazılım Verisi Kullanan Eklentilerin Gerçeklenmesi

Yazılım açık noktalarını arayüzler ile tanımlayacak, buna ek olarak iç verilerini de eklentilere açacaktır. Böylece bir eklenti hem yazılımda bir boşluğu dolduracak, hem de yazılımda çalışma zamanında oluşacak verileri de kullanarak güncelleme yapabilecektir.

```
public class ASembolojiEklentisi implements
EklentiArayuzu, MenuGuncellemeArayuzu {
public void baslat(){
  // yazılımda değişiklik yapılır.
  sembolojiGuncelle();
}
public void durdur(){
}
Public void sembolojiGuncelle (){
  ...
}
// yazılım tarafından gerekli menü elemanını almak için
// çağrılan metoddur.
public JMenuItem menuElemanıAl (){
...
}
}
```

Örnek bir eklenti kodu ise yukarıda verilmektedir. Burada eklenti bir yandan yazılımdaki menüye bir eleman eklerken, diğer yandan mevcut sembolojiyi yenisi ile değiştirmektedir.

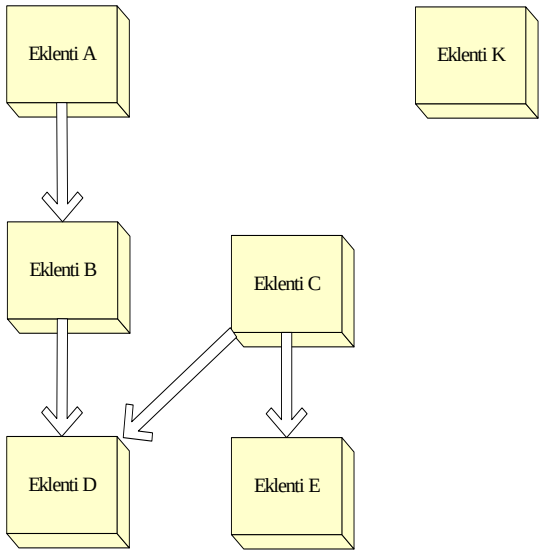
2.2.5. Eklentiler Arası Bağımlılık

UYGAR bileşeni eklentileri çalıştırmadan önce, eklentiler arası bağımlılıkları kontrol etmektedir. Birbirlerine bağlı eklentiler olabilmekte ve bu eklentileri yazılıma dahil etmek için doğru sırada çalıştırmak gerekmektedir. Eklentiler arası bağımlılığı tanımlamak amacıyla eklenti paketi içerisindeki tanım dosyası kullanılmaktadır.

```
Manifest-Version: 1.0
Eklenti-Sinifi: AraziModellemeEklentisi
Eklenti-Bagimliliklari: YukseklikVeriOkumaEklentisi
```

Yukarıdaki tanım dosyasına göre önce *YukseklikVeriOkumaEklentisi* çalıştırılmalı, şayet sorunsuz çalıştırılabildiyse *AraziModellemeEklentisi* çalıştırılmalıdır. Bu amaçla UYGAR bileşeni eklentileri çalıştırmadan önce eklentiler arası bağımlılıkları çözümlemekte ve eklenti bağımlılık ağacı oluşturmaktadır.

Örnek bir eklenti bağımlılık ağacı Şekil 5'te verilmiştir.



Şekil 5. Örnek Eklenti Bağımlılık Ağacı

Şekil 5'teki eklenti bağımlılıkları ele alındığında, Eklenti C'yi çalıştırmak için Eklenti E çalışıyor olmalıdır. Eklenti K bağımsız bir eklentidir ve diğer eklentiler ile çalıştırılma sırası fark etmemektedir.

UYGAR bileşeni, eklenti ağacını kullanarak eklentileri doğru sırada çalıştırmaktadır. Öncelikle hiçbir eklentiye bağlı olmayan eklentileri, daha sonra bağımlı oldukları eklentileri başarıyla çalıştırılabilmiş eklentileri çalıştırmaktadır. Ayrıca döngüsel bağımlılıklar da ele alındığından sorunlu eklentiler çalıştırılmayacak ve yazılım kararlı bir durumda tutulacaktır.

3. Eklenti Mimarisi Kullanımının Değerlendirilmesi

Eklenti mimarisi farklı projelerde kullanılmış ve edinilen tecrübeler uyarınca avantajları ve dezavantajları belirlenmiştir. Aşağıda alt başlıklar halinde avantajlar ve dezavantajlar anlatılmaktadır.

3.1. Eklenti Mimarisi Kullanmanın Avantajları

3.1.1. Kolay Genişleyebilirlik

Herhangi bir uygulama geliştirirken, başlangıçta tüm gerekleri tahmin etmek mümkün değildir. Bu yüzden geliştirilecek yazılım, değişikliklere ve yeni yeteneklerin eklenmesine açık olmalıdır. Örneğin, bir harita uygulamasında tüm harita formatlarını

destelemek mümkün değildir. Her yeni format desteği gerektiğinde, tüm yazılımı derleyip yeni bir yazılım sürümü oluşturmak tercih edilen bir yaklaşım değildir. Bunun yerine yazılıma yeni formatları desteklemek amacıyla eklemeler yapılabilmelidir. Eklenti mimarisini kullanan bir yazılımda geliştirici, desteklemek istediği veri formatını okuyan, işleyen ve altyapının istediği formatta uygulamaya dahil eden bir eklenti geliştirmekte ve yazılıma ekleyebilmektedir.

3.1.2. Uyumlanabilirlik

Büyük çaplı uygulamalarda çok fazla kaynak ve uyumlama dosyaları bulunmaktadır. Çoğu zaman bir projeye, uygulamadaki yeteneklerin tamamı yerine bir kısmının aktarılması istenmektedir. Bunu gerçekleştirmenin bir yolu, uyumlama dosyaları kullanarak yazılım yeteneklerinin ayarlanabilmesidir. İstenmeyen yetenekler uyumlama parametreleri kullanılarak ayarlanmaya çalışılmaktadır. Fakat bu hem zor hem de zahmetli bir iş olmaktadır. Her bir uyumlama değerinin kararlılığı test edilemeyebilir. Eklenti mimarisi, bu problemi ilgili yeteneklerin toplandığı eklentiler sayesinde kolaylıkla çözmektedir. Bu kullanıma en uygun örnek, Eclipse yazılım geliştirme aracıdır. Bu araçta istenilen eklentiler yüklenerek hedeflenen geliştirme ortamı oluşturulmaktadır. [2]

3.1.3. Çalışma Zamanı Güncellemelerinin Kolay Yapılabilmesi

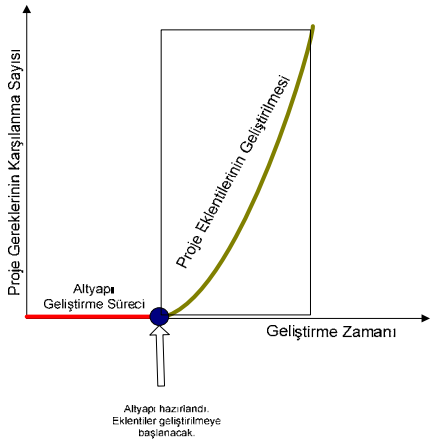
Özellikle sürekli çalışması gereken kritik sistemlerde güncelleme yapmak amacıyla sistemi kapatmak tercih edilmemektedir. Eklenti mimarisi bu problemi çözerek çalışma zamanında yazılıma eklenti yüklenebilmesini ve yazılımdan eklenti çıkarılmasını mümkün kılmaktadır. Güncelleme için yazılımın kapatılıp tekrar açılmasına gerek olmamaktadır.

3.1.4. Projeler İçin Hızlı Uygulama Geliştirme

Yazılıma zaman içinde eklemeler ve güncellemeler yapmak gerekmektedir. Yazılımın tüm kaynak kodlarına egemen birisi için değişiklik zamanı daha kısa, değişiklik süreci daha sancısız olsa da genellikle yazılımda güncelemeler yapan kişi ile yazılımı geliştiren kişi aynı olmamaktadır. Geliştirilen mimari ile farklı kişiler altyapının detaylarını bilmeden, sadece eklenti mekanizmasını öğrenerek yazılımda değişiklikler yapabilmektedirler.

Ayrıca büyük bir yazılım ancak farklı geliştiricilere bölünerek geliştirilebilmektedir. Eklenti mimarisi her bir eklentinin firma içinde ya da altyüklenici firmalar kullanılarak paralel olarak geliştirilebilmesini sağlamaktadır.

Şekil 6'da eklenti mimarisini kullanan bir yazılımın geliştirme süreci grafiği gösterilmektedir. Proje gereklerini gerçeklemeden önce, yazılımın eklentilerce genişleyecek açık noktalarını tanımlamak için harcanan süre az olmamıştır. Bu altyapı tamamlandıktan sonra harcanan süre projelerin gereklerine yönelik olmuştur. Her proje için harcanan süre grafiğin 2. bölümündedir. Böylece 1. bölüm sadece bir kez o da yazılım geliştirme sürecinin ilk başında harcanmıştır.



Şekil 6. Eklenti Mimarisine Sahip Uygulama Geliştirme Süreci

3.1.5. Sadece Kullanılacak Yetenek ve Kaynak Kodların Teslim Edilmesi

Yazılım sadece bir projenin ihtiyaçlarını karşılayan eklentiler ile teslim edilmekte, böylece gereksiz kod ve işlevler teslim edilmemektedir. Bu da yazılımın koşacağı bilgisayardaki kaynak kullanımı daha verimli hale getirmektedir.

3.1.6. Altyapıda Kontrollü Güncelleme Yapılması

Eklentiler altyapıya kontrollü olarak erişebilmektedirler. Eklentiler belirli arayüzler üzerinden yazılım ile haberleşmekte ve veri alışverişinde bulunmaktadırlar. Bu da yeni eklentilerin sisteme zarar vermesini engellemekte, sadece izin verilen güncellemelerin kontrollü olarak yapılabilmesine olanak sağlamaktadır.

3.1.7. Sadece Güncellenen ve Eklenen Yeteneklerin Test Edilmesi

Eklentilerin yazılıma kontrollü erişimleri sayesinde yazılımın her teslimatta test edilmesine gerek kalmamaktadır. Sadece yeni eklenen eklentilerin test edilmesi yeterli olmaktadır. Örneğin menüye yeni bir eleman eklenmesi, menünün tamamen dışarıya verilmesi ile değil, menüye eleman ekleyen metotların sunulması ile gerçekleştirilmiştir. Bu sayede geçerli olmayan bir değişikliğin yapılmasına izin verilmemiştir. Böylece her yeni eklenti için temel yazılım testlerinin tekrar yapılmasına gerek olmamıştır.

3.2. Eklenti Mimarisi Kullanmanın Zorlukları

3.2.1. Geliştirme Dili Yetenekleri

Eklenti mimarisine sahip bir uygulama geliştirmek için seçilecek geliştirme dili önemlidir. Mimarinin çalışma zamanı avantajları, seçilecek dilin çalışma zamanı desteği ile yakından ilgilidir. Örneğin java programlama dilini kullanmanın birçok avantajı olmuştur. Java, yapısı gereği çalışma zamanı byte kodlarını kullanarak sınıf yüklemesi yapabildiğinden [4], çalışma zamanında eklentilerin çalıştırılması ve durdurulması kolay olmaktadır.

3.2.2. Eklenti Tanım Bilgilerinin Güncel Tutulması

Eklentilerin tanım dosyalarının güncel tutulması son derece önemlidir. Bu dosyalarda çalışma zamanında kullanılacak ve yazılımın eklentiler hakkında önemli kararlar almasını sağlayacak bilgiler bulunmaktadır. Ayrıca yazılımın ve eklentilerin uyumluluğunun da takip edilmesi tanım dosyalarının güncel ve doğru olması ile bağlantılıdır. Bu dosyalardaki bilgilerin yanlışlığı eklentilerin çalıştırılamamasına sebep olmaktadır.

3.2.3. Eklentilerin Dokümantasyonu

İstenilen yetenekleri elde etmek için bazı eklentilerin yazılıma eklenmesi gerekmektedir. Burada dikkat edilmesi gereken eklentilerin bağımlılıkları konusudur. Bir eklentinin çalışabilmesi için varsa bağımlı olduğu diğer eklentilerin yüklü olması gerekmektedir. Bu lego parçalarının bir araya getirilerek bir oyuncak oluşturulmasına benzemektedir. Hangi parçayı seçmek, hangi parçalarla birleştirmek, sonuçta sürpriz yaşamamak açısından önemlidir. Bu yüzden eklentilerin yetenekleri ve sundukları servisler için detaylı dokümanlar hazırlanması gerekmiştir. Doküman eksikliği sorunlara sebep olmuştur.

3.2.4. Geliştirme Zamanı Versiyon Takibi

Temel yazılım ve her eklenti farklı projeler olarak geliştirilebilmektedir. Bu da kodların yapılandırma yönetimini zorlaştırmaktadır. Her yazılım projesinin ayrı bir sürümü olabilmektedir. Versiyonların uyumluluğunun iyi takip edilmemesi, yazılım ve eklentilerin çalışma zamanında uyumsuzluğuna yol açmaktadır. Bunu engellemek için her eklentinin uyumlu olduğu temel yazılımın ve diğer eklentilerin sürümleri eklenti tanım dosyalarında tanımlanmıştır.

4. Sonuç

Bu bildiride, eklenti mimarisi ve bu mimariyi desteklemek amacıyla geliştirilen UYGAR bileşeni anlatılmıştır. Eklenti mimarisinin uygulama yazılımlarında kullanımı, bir uygulama yazılımın farklı eklentiler ile zenginleşmesini sağlamıştır. Ayrıca eklentilerin tekrar tekrar kullanılabilirliği de zaman ve iş gücü kaybını engellemiştir.

Eklenti mimarisi, temel yetenekleri içeren bir yazılım, genişlemeyi sağlayacak UYGAR bileşeni ve eklentilerin ortaya çıkmasını sağlamıştır.

Bildiride eklenti mimarisinin kullanımı, UYGAR bileşeni, eklentilerin nasıl gerçekleneceği ve eklenti mimarisinin değerlendirilmesi anlatılmıştır.

5. Kaynak

[1] Michael Pilone, *Plugins & Java*, Dr. Dobbs Jounal, December 2004

[2] The Eclipse Project, www.eclipse.org, Son Erişim: October 2006

[3] Szyperski, C., *Component software: Beyond object oriented programming*, Addison-Wesley Pub Co, 1997

[4] Liang, S., Bracha, G., *Dynamic Class Loading in the Java Virtual Machine*, 1998

[5] Sun Microsystems Inc., http://www.sun.com/Java, Son Erişim: October 2006

YENİDEN KULLANILABİLİR BİLEŞEN GELİŞTİRME YÖNTEMİ

Özgü Özköse ERDOĞAN
ASELSAN A.Ş.
ozkose@mst.aselsan.com.tr

Baki DEMİREL
ASELSAN A.Ş.
demirel@mst.aselsan.com.tr

Alper BOSTANCI
ASELSAN A.Ş.
bostancı@mst.aselsan.com.tr

Öz

ASELSAN MST Grubu Radar ve Elektronik Harp projeleri için bir referans mimari model tanımlanmıştır; bu model ortak ihtiyaçları karşılayan yeniden kullanılabilir bileşenleri içermektedir.

Bu bileşenlerin geliştirilmesinde çevik yöntem kullanımı, ortak geliştirme yapılması, her bir bileşenin yazılım ürün hattı olarak takip edilmesi yöntemleri benimsenmiştir. Bileşenlerin geliştirilmesinde referans mimaride de güncellemeler yapılması gerekebilmiştir ve gerekli güncellemeler yapılmıştır. Uygulanan bu yöntem ile referans mimari daha da olgunlaşmaktadır.

Uygulama sonuçları başarılı olmuş, bileşenlerin aktif projelerde kullanımıyla referans mimari de kullanılmaya başlanmıştır.

Bu bildiride bileşenlerin geliştirilmesinde kullanılan yöntem örneklerle anlatılarak genel değerlendirme yapılmıştır.

1. Giriş

ASELSAN yaklaşık 20, MST Grubu ise 15 yıldır Radar ve Elektronik Harp (REH) sahasında faaliyet göstermektedir. ASELSAN'ın mevcut ve yeni projeleri incelendiğinde geliştirmekte olduğu ürünlerin büyük oranda ortak yetenekler ile farklı platformlarda sunulduğu görülmektedir.

ASELSAN'ın pazarda varlığını ve iddiası sürdürebilmesi için daha ucuz, yeni ihtiyaçlara hızla uyumlandırılarak daha kısa sürede teslim edilebilen ürünler geliştirmesi gerekmektedir.

Bu durumda her projeyi baştan tasarlamak yerine, her projenin ihtiyacına göre ölçeklendirilebilen ve uyumlanabilen tasarımlar ortaya koymak gerekmektedir; böyle bir tasarım ise ortak ihtiyaçların tanımlı bir mimari ile yeniden kullanılabilir bileşenler olarak geliştirilmesi ile sağlanabilir.

ASELSAN MST Grubu'nda, Radar ve Elektronik Harp projeleri için bir mimari model oluşturulmuş ve bu mimari içinde ortak ihtiyaçları adresleyen bileşenler belirlenmiştir, bu mimariye Radar Elektronik Harp Fonksiyonel Referans Modeli (REFoRM) adı verilmiştir. Halen, tanımlanan bileşenler, yönetilen bir yetenek setini paylaşan ve ortak çekirdek yapılardan, tanımlanmış yöntemlere göre geliştirerek "Yazılım Ürün Hattı" olarak ele alınmaktadır.

Yazılım Ürün Hattı olarak ele alınan bileşenler aşağıdaki dört adımdan oluşan yöntemle geliştirilmekte, kullanıma alınmakta ve yaşam döngüsüne girmektedir (Bknz. Şekil 1):

- İhtiyaçların toplanması
- Geliştirme ekibinin kurulması
- Çevik yöntemle bileşenin geliştirilmesi
- Bileşenin kullanıma alınması

Bildiride "bileşen" terimi REFoRM kapsamında tanımlı bileşenleri ifade etmektedir.

2. Yöntem

REFoRM içinde yer alan bileşenler aktif projelerdeki kullanım ihtiyaçları göz önüne alınarak önceliklendirilir. Bileşenler önceliğine ve gerektirdiği iş yüküne göre seçilerek geliştirilmeye başlanır.

2.1 İhtiyaçların Toplanması

Geliştirilecek olan bileşenin ihtiyaçlarının belirlenmesi amacıyla, ilgili konuda uzman elemanlar toplanarak geliştirilmiş, geliştirilmekte ve teklif aşamasında olan projelerin bu konuda bilinen ihtiyaçlarını beyin fırtınası yöntemiyle serbestçe önerirler. Öneriler incelenerek birbiriyle ilişkili olanlar gruplanır ve çelişenler elenir.

Bu aşamada bileşeni takip edecek müşteri ekibi de belirlenir. Müşteri ekibi geliştirilecek bileşenleri müşteri bakış açısıyla değerlendiren ve çevik yöntemde gelişme adımlarını takip eden ekiptir; bu ekip farklı projelerde çalışan uzman elemanlardan oluşur. Böylelikle müşteriyi merkeze alarak üretilen ürünün mimarinin ve projelerin ihtiyaçlarına uygun olması garanti altına alınır.

Müşteri ekibi, daha önceden beyin fırtınası yöntemiyle belirlenip ortaya konan ihtiyaçları değerlendirerek, bu ihtiyaçları hikayeler olarak yazar; her bir hikaye tam çalışan tek bir ihtiyacı tanımlar.

Müşteri ekibi ihtiyaçları değerlendirirken bileşenin REFoRM içindeki yerine ve tanımına uygun olmasına dikkat eder, uygun olmayan ihtiyaçlar hikaye olarak yazılmaz, REFoRM kapsamında yeniden değerlendirilmek üzere derlenir.

Bu hikayelerden hareketle yetenek ağacı oluşturulur. Yetenek ağacı, ilgili bileşenin sahip olduğu yetenekleri, yeteneklerin niteliklerini (zorunlu, seçmeli, değişken) ve değişimlerini izlemek amacıyla kullanılır.

2.2 Geliştirme Ekibinin Oluşturulması

Bileşen geliştirme ekibi, değişik projelerde çalışan elemanlardan oluşturulur. Ekip içerisinde tasarım tecrübesine sahip olan en az bir elemana ek olarak değişik tecrübe seviyelerinde elemanlar bulunur. Ekip elemanlarının iş gücünün büyük bir bölümünü bileşen geliştirilmesi için kullanması beklenir.

Karma bir ekip kurmanın

- geliştirilen bileşeni sahiplenmenin arttırılması,
- mimarinin öğrenilmesi ve sahiplenilmesi,
- değişik ekiplerdeki kültürlerin paylaşılması

açılarından faydaları vardır.

Bileşenin sahiplenilmesinin artması proje ekipleri tarafından kullanılmasını ve bakımın paylaşılmasını garantiler.

Bileşen geliştirme ekibi, bileşenin genel mimari içindeki yerini bildiği için mimariyi de sahiplenerek projesinde kullanılmasını ve mimariye öneriler vererek de daha olgun bir mimarinin oluşmasını sağlar.

Farklı projelerde oluşan kültürler kaynaşarak ortak bir noktaya ulaşılır; böylece çalışanların birbiri ile iletişimi artarak ortak mimari ve bileşenlerin kullanımı yaygınlaşır.

2.3 Çevik Yöntemle Bileşenin Geliştirilmesi

Çevik yöntemle bileşen geliştirilmesi, bu yöntemin;

- değişen gerekleri değişen dünyanın bir parçası kabul ederek geliştirilen bileşenin yeni gereklere daha çabuk uyumlanabilir olmasını sağlaması,
- sadece o anda ihtiyaç duyulan yetenekleri adresleyerek bileşende basit bir tasarımı zorlaması,
- müşteri bakış açısıyla hazırlanan hikayelerdeki ilerlemelerin görülebilmesini sağlaması,
- her zaman çalışan bir bileşen sağlaması

nedenleriyle seçilmiştir.

Geliştirme ekibi hikayelere puan verir; puanlar en küçük iş referans alınarak hikayeler arası oranı gösterecek şekilde ölçeklendirilir; puan verirken müşteri ile iletişim devam eder; net anlaşılmayan hikayeler görüşülür, bazı hikayeler birleştirilerek puan verilebilir, birden fazla işlem içeren ve/veya puanı yüksek hikayeler parçalanabilir.

Geliştirme sırasında projenin durumunu ve ne zaman biteceğini gösteren “kalan puan” (burn-down), “biten puan” grafikleri kullanılır; bu grafiklerin kullanım amacı geliştirme ekibini kontrol etmek değil, durumu izlemektir.

Müşteri hikayeleri bileşenleri ilk kullanacak olan projelerin ihtiyaçlarına, hikayelerin bağımlılıklarına ve puanlarına göre önceliklendirir.

Müşteri takibinin yapılacağı iterasyon süresi belirlenir, iterasyon süresinin gün/hafta bazında olması beklenir.

Geliştirme ekibi ilgili iterasyon için istediği toplam puanı belirtir, müşterilerin öncelikli hikayeleri farklı olabilir, müşteri ekibi kendi arasında anlaşarak ilgili iterasyonda gerçeklenmesini istediği hikayeleri toplam puan çerçevesinde belirleyerek geliştirme ekibine verir ve iterasyon başlamış olur.

Geliştirme ekibi hikayeleri ekip içindeki uzmanlık seviyelerine ve hikayenin zorluk derecesine göre paylaşır. Ekip gerçeklemeye başlamadan önce ortak bir yazılım tasarımı yapar, ardından kodlamaya geçer.

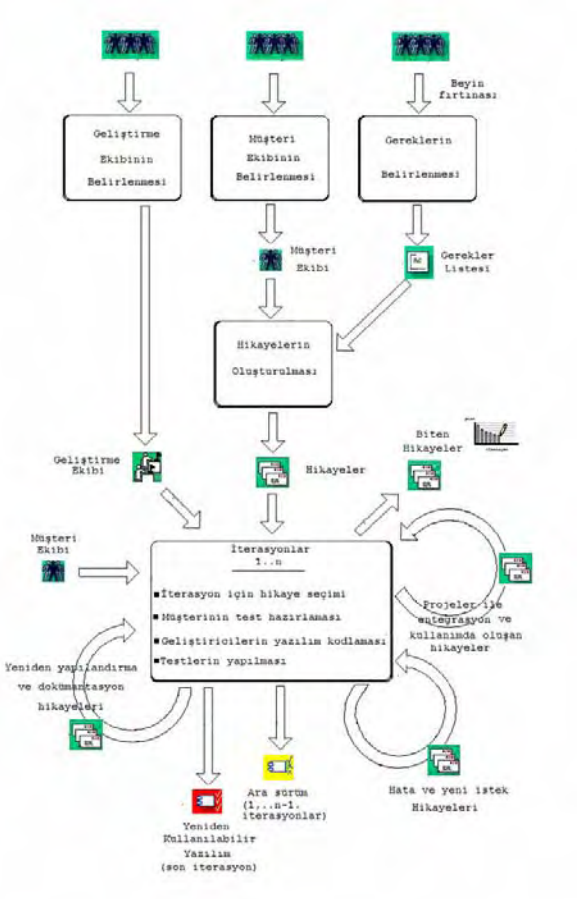
Geliştirme ekibi elemanları sorumlu oldukları hikayeleri gerçeklemeye başlarlar; test öncelikli geliştirme ve değişik tecrübe seviyesindeki geliştiricilerin aynı seviyeye gelebilmesi için eşli programlama önerilir.

Geliştirme sırasında sadece verilen hikayeler gerçeklenir, çalışacak en basit tasarım sağlanır, hikayelerde anlaşılmayan, kararsız kalınan noktalar müşteriye danışılarak belirsizlikler giderilir.

Müşteriler ilgili iterasyonda seçilen hikayeler için otomatik test araçları ile kabul testlerini yazarlar. Hikayeler ne yapılması gerektiğini tariflerken, kabul testleri ihtiyacın nasıl yerine getirileceğini tanımlayarak hikayeleri tamamlar. İlgili hikaye için hatalı durumların kontrolü için de kabul testleri hazırlanır.

Kabul testleri iterasyonun ortasında geliştiricilere aktarılır, kabul testlerinde anlaşılmayan veya tasarım ile uyuşmayan konular hemen müşteri ile görüşülür. Kabul testleri geliştiricinin yazılım mimarisini belirlemesinde önemli rol oynar.

Geliştirme ekibi hikayeleri gerçeklemenin yanında, kabul testlerinin de yapılabilmesi için kodlama yapar. Geliştirme sürecinde kabul testleri ve uygulanıyorsa birim testleri çalıştırılır.



Şekil 1. İş Akışı

Müşteri ve geliştiricilerin katıldığı ortak toplantıda değerlendirme yapılmasıyla iterasyon sonlanır. Bu toplantıda her hikaye için;

- kabul testlerinin geçip geçmediği görülür,
- çalışan yazılım gözlenir,
- hikayelerin gerçeklenmesinde müşteri isteği ile uyumsuzluk veya hatalar varsa hata hikayeleri olarak eklenir
- biten hikayeler işaretlenir,
- yeni ihtiyaçlar doğrultusunda yeni hikayeler yazılabilir, ya da varolan bazı hikayeler iptal edilebilir,
- geliştirme ekibinden gelen yeniden yapılandırma ve doküman hazırlama hikayeleri eklenebilir,
- yeni hikayelere puan verilir.

İterasyon sonunda kalan puan ve biten puan grafikleri güncellenir.

Yeni iterasyon puanın belirlenmesi ve iterasyon hikayelerinin geliştirme ekibine verilmesi ile başlar, tüm hikayeler bitirilinceye kadar iterasyon döngüsü devam eder.

Baştaki iterasyonlarda üretkenliği gösteren iterasyonda tamamlanan toplam hikaye puanı belirli bir süre dalgalanır. Geliştirme ekibin birbirine alışması, yazılım mimarisinin belirginleşmesi, kullanılan yöntemin benimsenmesi ile dalgalanma azalır ve belirli bir ortalama seviyeye erişilir.

Bileşenin ara sürümleri çıkarılarak projelerde kullanıma alınması sağlanır. Böylece bileşenin gerçek kullanım ortamına entegrasyonu sağlanarak yeni ihtiyaçlar, değişiklikler ve hatalar erken aşamada görülür; geri beslemelerin yeni hikayeler şeklinde sonraki iterasyonlarda ele alınması sağlanır. Ara sürümlerde entegrasyonu sağlayacak seviyede dokümantasyon da sağlanır.

2.4 Bileşenin Kullanıma Alınması

Tüm hikayeler tamamlandıktan sonra proje entegrasyonu yapılarak hata olmadığı görülür, içinde dokümanlarıyla birlikte çalıştırılabilir bileşenin olduğu ürün paketi hazırlanır, bileşen çalışmasının tamamlandığı duyurulur, yeni istekler ve hataların değişlik sistemi üzerinden takip edildiği bakım dönemi başlar.

3. Örnek Bileşen: Seyrüsefer

Seyrüsefer bileşeni; zaman, üç boyutlu koordinat, oryantasyon ve manyetik sapma gibi seyrüsefer bilgilerinin değişik donanım kaynaklarından elde edilmesini, ayarlanabilmesini ve kullanıcı uygulamalara dağıtılmasını sağlayan bir kütüphane yazılımıdır.

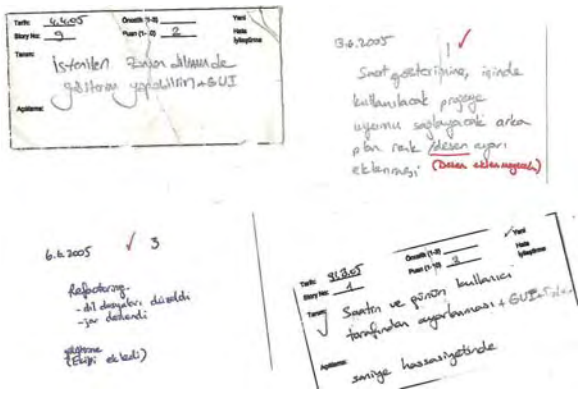
Seyrüsefer bileşeninin geliştirilmesine “Saat bileşeni” adı ile başlanmıştır, ancak geliştirme sırasında çıkan ihtiyaçlar REFoRM kapsamında değerlendirilerek bileşenin Seyrüsefer olarak tanımlanmasının ve REFoRM’un güncellenmesinin uygun olacağına karar verilmiştir.

3.1 Seyrüsefer Bileşeni İhtiyaçlarının Toplanması

Seyrüsefer bileşeninin ihtiyaçları, değişik projelerin yazılım yöneticilerinin ve projelerde daha önce bu konuda geliştirme yapmış deneyimli tasarımcıların oluşturduğu on kişilik bir ekiple beyin fırtınası yöntemiyle toplanmıştır. İlk aşamada bu bileşenin sadece zaman bilgisini işlemesi öngörülmüştür.

Değişik projelerde gerçek müşteri ile temasta bulunan, onların rolünü üstelenebilecek, REFoRM’un yapısını da iyi derecede bilen yazılım yöneticilerinden oluşan üç kişilik bir grup müşteri ekibi olarak belirlenmiştir.

Müşteri ekibi daha önceden oluşturulmuş olan ihtiyaçları kullanarak ve Seyrüsefer bileşenin REFoRM içindeki yerini göz önüne alarak, müşteri bakış açısıyla hikayeler hazırlamışlardır. Bu ihtiyaçlar gruplanmış ve toplam 15 hikaye yazılmıştır. Şekil 2'de hikaye örnekleri verilmiştir.



Şekil 2. Hikaye Örnekleri

Bu hikayelerden Seyrüsefer bileşeni için yetenek ağacı oluşturulmuştur (Bknz. Şekil 8).

3.2 Seyrüsefer Bileşeni Geliştirme Ekibinin Oluşturulması

Üç değişik projeden değişik deneyim seviyelerinde üç kişilik bir geliştirme ekibi belirlenmiştir, bu ekipten bir kişi daha önceki projelerinde seyrüsefer bilgilerini değişik donanımlardan elde eden yazılımlar hazırlanmasında çalışmıştır. Geliştirme ekibinin geliştirme çalışmaları öncesinde REFoRM hakkında bilgi sahibi olması da sağlanmıştır.

3.3 Çevik Yöntemle Seyrüsefer Bileşeninin Geliştirilmesi

Geliştirme ekibi, daha önceden yazılmış olan hikayelerin tümünü inceleyerek hikayelere puan vermiştir. Puanlamada 1 ile 5 arasında puanlar kullanılmıştır, puanlama sırasında anlaşılamayan hikayelerle ilgili müşteriden bilgi alınmıştır.

Puanlanmış hikayeler Şekil 3'de gösterilen puan/iterasyon tablosuna aktarılmıştır.

no	hikaye	iterasyon	puan	tarih	durum	açıklama
15	saat uygulaması her ortamda çalışır	1	1	04.04.05	b	işletim sisteminden bağımsız
1	saatin ve günün kullanıcı tarafından ayarlanması	1	3	31.03.05	b	GUI + takvim çikacak, saniye hassasiyetinde
2	saatin ve tarihin devamlı gösterilmesi	1	3	31.03.05	b	saat saniye hassasiyetinde, gün: dd/mm/yy

Şekil 3. Puan/İterasyon tablosu

Müşteri ve geliştirme ekibi iterasyon süresini bir hafta olarak belirledikten sonra geliştirme ekibinin ilk iterasyonun puanını belirlemesi ve müşteri ekibinin de ilk hikayeleri seçmesi ile geliştirme başlamıştır.

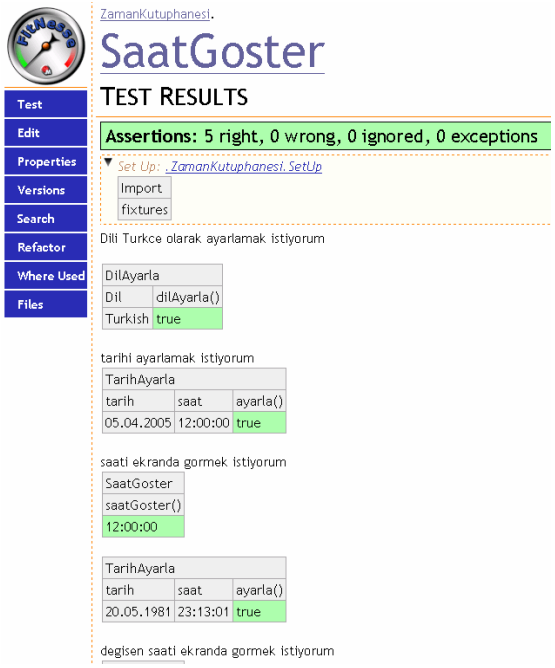
Geliştirme ekibi hikayeleri paylaşarak, hikayelerin nasıl geliştirileceği konusunda değerlendirme yaptıktan sonra iterasyon süresince eşli programlama ve test öncelikli geliştirme yapmıştır. Öncelikle hikayelere karşılık gelen işlevler için iş paylaşımı yaptıktan sonra bu işlevlere özgü birim testlerini hazırlayıp, bu testleri karşılayan yazılımı kodlamışlardır.

Müşteri ekibi düzenli olarak kabul testlerini hazırlamış, böylece hikayelerden beklentilerini daha detaylı olarak geliştirme ekibine aktarabilmiştir. Şekil 4'te örnek kabul testleri gösterilmektedir.

Şekil 4. Müşteri kabul testleri

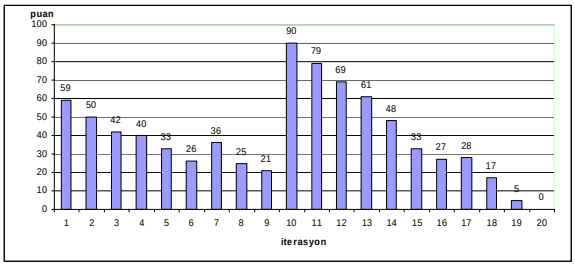
Kabul testleri iterasyon ortasında geliştirme ekibine aktarılmaya çalışılmıştır. Geliştirme ekibinin kabul testlerini incelemesi ile bazı hikayelerin yanlış yorumlanabildiği görülmüş ve kodlama beklentileri karşılayacak şekilde güncellenmiştir. Geliştirme açısından çok zor olabilecek beklentiler iki ekip tarafından görüşülerek bazen de kabul testleri güncellenmiştir.

İterasyon toplantılarında Şekil 5'de gösterilen müşteri testlerinin başarıyla geçtiği görülmüş, çalışan yazılım gözlenmiştir.

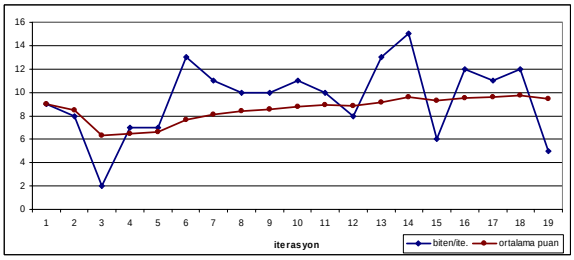


Şekil 5. Müşteri kabul test sonuçları

Her iterasyon sonunda bileşenin yeni yetenekler eklenmiş yeni bir versiyonu çıkmıştır. İterasyon sonunda kalan puan ve biten puan/iterasyon grafikleri güncellenerek ilerleme izlenmiştir, Bknz. Şekil 6.



Şekil 6. Hikayeler Kalan Puan Grafiği

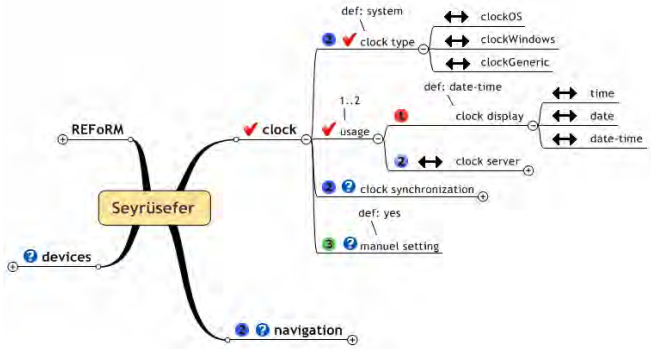


Şekil 7. Biten Puan - Ortalama Puan / İterasyon Grafiği

Geliştirme ekibin yazılım mimarisini tasarlaması ile zaman içinde verimliliği artmıştır ve Şekil 7'de görüldüğü üzere iterasyonlardaki tamamlanan toplam hikaye puanları zaman içinde artarak ortalama on puan seviyesinde gerçekleşmiştir.

Yedinci iterasyonda dokümantasyon ve yazılım yeniden yapılandırması hikayeleri eklenmiştir.

Dokuz iterasyon sonrasında müşteri ekibi, bu bileşenin zaman bilgisini işlemesine ek olarak seyrüsefer bilgilerini de işlemesine karar vererek kapsamını genişletmiş ve ek hikayeler hazırlamıştır. Şekil 6'da dokuzuncu iterasyon sonrasında 69 puanlık on sekiz hikayenin eklenmesinin etkisi görülmektedir. Bu aşamada Şekil 8'de gösterilen yetenek ağacı da güncellenmiştir.



Şekil 8. Yetenek Ağacı

Tüm geliştirme süresince toplam 20 iterasyonda, 185 puanlık 82 hikaye gerçeklenmiş, 30 puanlık 10 hikaye iptal edilmiştir.

Ara iterasyonlarda;

- geliştiricilerin isteği üzerine 3 tane toplam 10 puanlık yeniden yapılandırma hikayesi,
- karşılaşılan hataların düzeltilmesi için toplam 10 puanlık 7 hata hikayesi,
- doküman hazırlanması amaçlı toplam 28 puanlık 11 hikaye

eklenmiştir.

6. ve 16. iterasyonlarda iki ayrı sürüm iki değişik projede kullanılmıştır, bu projelerle entegrasyon için toplam 9 puanlık iki hikaye eklenmiştir. Bileşenin projelere entegrasyonu ile bileşenin proje içinde kullanımı doğrulanmıştır.

3.4 Seyrüsefer Bileşenin Kullanıma Alınması

Seyrüsefer bileşenin geliştirilmesi tamamlandığında içinde dokümanlarıyla birlikte çalıştırılabilir bileşenin olduğu son ürün paketi hazırlanarak 01.01 sürümü çıkartılmış, duyurulmuş ve tanıtımı yapılmıştır. Bu dönemde yeni istekler ve hataların düzeltilmesi yapılmaktadır.

Seyrüsefer bileşeninin ilk sürümü üç projede kullanıma alınmıştır.

3.5 Seyrüsefer Bileşeni Geliştirmenin Genel Değerlendirmesi

Seyrüsefer bileşeni yedi adam-ay işgücüyle geliştirilmiştir. Bileşenin yeni projelerde kullanıma alınması yarım günden kısa sürmüştür. Eski projelerde en az dört adam-ay gerektiren yetenekler Seyrüsefer bileşeninin kullanılmasıyla bir buçuk adam-gün işgücüyle üç projeye eklenmiştir, ilgili projeler yarım gün içinde ciddi bir iş gücü gerektiren yeteneklere dokümanı hazırlanmış ve test edilmiş olarak sahip olmuşlardır.

REFoRM kapsamında geliştirilen bileşenin kullanımı ile en az beş adam-ay işçilik kazanılmıştır, yeni projelerdeki kullanımlar ile işçilik kazanımı daha da artacaktır. Sonuç olarak projelerdeki ortak ihtiyaçların aynı şekilde karşılanması ile birlikte bu yeteneklerin daha kaliteli ve daha ucuza teslim edilmesi sağlanmıştır.

4. Kazanımlar

Günümüzde pazar ihtiyaçlarını hızlı, kaliteli ve daha az maliyetle karşılayabilmek için yazılımların yeniden kullanımı çok önem kazanmıştır. Bu ihtiyaç geliştirilen projelerin ortak bir mimari yapıya oturtulması ve mimari yapı içinde yer alan ana bileşenlerin ortak olarak geliştirilmesi ile sağlanmaktadır.

Eski projelerde de proje ekipleri kendi ihtiyaçlarını kütüphane olarak hazırlamışlardır, ancak bu kütüphaneler genel kullanım açısından başarılı olmamıştır. REFoRM kapsamında geliştirilen bileşenler geliştirilmekte olan projelerde aktif olarak kullanılmaktadır.

Başarının nedenleri altı madde ile sıralanabilir:

1. Bileşenler çevik yöntemle müşteri odaklı geliştirildiği için doğru yetenekler içerdiği ve kalitesi garanti altına alınır.
2. Bileşenler karma bir ekip tarafından geliştirildiği için genel kabul görür ve doğru olarak kullanılır.
3. Bileşen ihtiyaçlarının bir proje ekibi tarafından belirlenmesi yerine farklı projelerden elemanlar ile oluşan bir ekip tarafından belirlenmesi, bileşenlerin gerçekten ihtiyaç duyulan yetenekleri içermesini sağlar.
4. Bir projede geliştirilen kütüphane başka bir projenin yazılım mimarisine uymayabilir. Ancak bileşenin tanımlı bir mimaride yerinin olması, her projede aynı şekilde kullanılmasını sağlar.
5. Bir proje kapsamında geliştirilen kütüphane genel yetenekler içerse dahi sadece o proje tarafından kullanılan yetenekleri test edilmiş olur. Bileşenlerin ise bütün yetenekleri tam olarak test edilir.
6. Bir proje ekibi tarafından geliştirilen kütüphane başka bir proje tarafından kullanıldığında yeni ihtiyaçlar oluşur, bu ihtiyaçlar her ekip tarafından sadece kendi versiyonuna eklenir; aslında bu durumda gerçeklenen yeniden kullanım değil, kopyala-ve-sahiplen'dir. Bileşenler Yazılım Ürün Hattı olarak takip edildiği için yeni yetenekler genel yapı düşünülerek eklenir ve her versiyonun hangi yetenekleri içerdiği doğru olarak takip edilir.

5. Sonuç

REFoRM tanımlandıktan sonra başlayan çalışmalar ile bir buçuk yıl içinde farklı boyutlarda beş adet bileşen geliştirilerek projelerde kullanıma alınmıştır. Bileşenlerin projelerdeki kullanımı, bileşenlerin iş ürünü olarak kalitesi, genel maliyeti ve müşteri memnuniyeti değerlendirildiğinde uygulanan sürecin şelale yönteminde tanımlanan "ihtiyaçları tanımla, son ürünü test et" yöntemine göre çok daha etkili olduğu görülmüştür.

ASELSAN MST Grubu, REFoRM ile tanımlanan bileşenleri aynı yöntem ile geliştirmeye ve REFoRM yapısını güncel tutmaya devam edecektir. Bu kapsamda 2 adet bileşen için geliştirme çalışması devam etmektedir, 2 adet bileşen için de ihtiyaçlar belirlenmiş ve diğer bileşenler için de planlama yapılmıştır.

Bileşenlerin geliştirilmesinde mimari tasarıma dayanılması, projelerin ortak ve özel ihtiyaçlarının görülebilmesi, bu ihtiyaçların Yazılım Ürün Hattı olarak yürütülmesi, geliştirmenin proje ekipleri tarafından ortak yapılması ve müşteri odaklı geliştirmeye önem verilmesi başarının anahtarlarıdır.

6. Kaynaklar

[1] Len Bass, Paul Clements, Rick Kazman, Software Architecture in Practice, Addison-Wesley, 1998.
[2] Beck, Kent "XP Immersion"
[3] Jan Bosch, Design & Use of Software Architectures – Adopting and Evolving a Product-line Approach, Addison-Wesley, 2000.
[4] ASELSAN MST Grubu, REFoRM – RADAR Elektronik Harp Fonksiyonel Referans Modeli, 2004

Yazılım Mimarileri Üzerinde "Eclipse" Etkileri: Komuta Kontrol Sistemleri Örnek Analizi

Tankut Koray
Mühendis, ASELSAN A.Ş. Mikrodalga ve Sistem Teknolojileri Grubu
tkoray@aselsan.com.tr

Özet

Eclipse [1]; "zengin istemci uygulamaları" olarak bilinen, açık kaynak kodlu ve platformdan bağımsız bir yazılım çerçevesidir. Bu çerçeve üzerinde, sadece yazılım geliştirme ortamları değil, birçok farklı amaca yönelik yazılımlar da geliştirilebilmektedir. Eclipse çerçevesinin yaygınlaşması ve kullanılmasıyla, yazılım mimarilerinde bileşen tabanlı ve servis tabanlı yaklaşımlar etkisini arttırmıştır. Bu yaklaşımlar, komuta kontrol sistemleri gibi farklı ve karmaşık işlevleri bir araya getiren sistemlerde tekrar kullanılabilir bileşenler ve mimariler ortaya çıkmasını sağlamaktadır.

ASELSAN'da geliştirilmekte olan komuta kontrol yazılımlarında kullanılmak üzere, bileşen tabanlı ve servis tabanlı yaklaşımlar uygulanarak ortak bileşenler ve ortak mimariler geliştirilmektedir. Bu bildiri kapsamında, ASELSAN komuta kontrol sistemlerinde kullanılması düşünülen genel yazılım mimarisi yaklaşımı ve bu yaklaşım çerçevesinde geliştirilen bileşenler özetlenecektir.

1. Giriş

Eclipse, birçokları için Java dili için bir yazılım geliştirme ortamı olmakla birlikte yazılım dünyasına getirdiği yaklaşımlar ve gerçeklemeler, birçok büyük şirketi etkilemiş ve birçok ürünün Eclipse tabanlı olarak yeniden tasarlanarak zengin özelliklerle karşımıza çıkmasına yol açmıştır.

İlk bölümde, Eclipse çerçevesi ve mimarisi incelenecek ve Eclipse'in bileşen yönetim altyapısını oluşturan OSGi (Open Services Gateway initiative) tanımı [2] özetlenecektir.

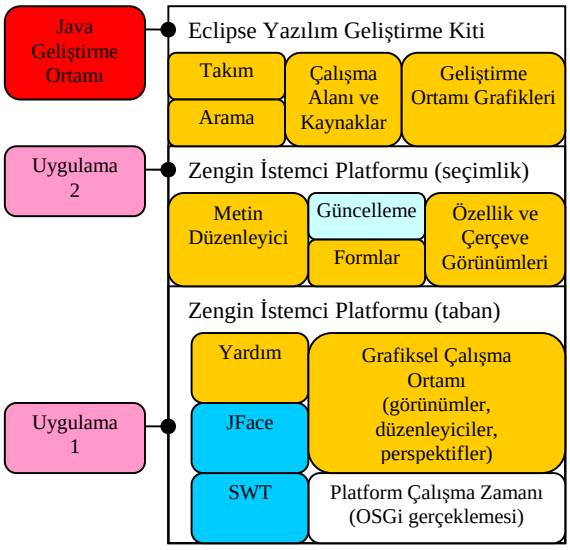
İkinci bölümde, geleneksel yazılım mimarileri ile bileşen tabanlı mimariler arasındaki farklar komuta kontrol yazılımları özelinde örneklerle açıklanacaktır.

Devam eden bölümlerde, ASELSAN'da geliştirilecek komuta kontrol yazılımlarında kullanılacak bileşen tabanlı mimarilerden ve bileşenlerden bahsedilecektir.

Sonuç bölümünde, bu mimarilerin ve bileşenlerin nasıl bir araya gelerek bir komuta kontrol yazılımı oluşturabilecekleri ortaya konulacaktır.

2. Eclipse Çerçevesi ve Mimarisi

Eclipse'in en önemli özelliklerinden biri, bileşen tabanlı tak-çıkar yaklaşımını çok iyi uygulaması ve işletebilmesidir. Eclipse tabanlı geliştirilen birçok başarılı yazılımın kanıtladığı üzere, bir yazılım tamamen tak-çıkar bileşenlerle yapılandırılarak daha işlevsel, daha hızlı ve daha güvenilir olabilmektedir.



Şekil 1 Eclipse Mimarisi

Şekil 1'de Eclipse mimarisinin genel bir görünümü sunulmuştur. Eclipse Platformu, içerisinde birçok hazır eklenti içermektedir. Bunların bir uygulama

geliştirmek için gerekli olan en ufak alt kümesi, Zengin İstemci Platformu olarak adlandırılmaktadır.

Eclipse'in temelinde bileşenleri bulan, yükleyen ve çalıştıran bir mimari vardır. Bu mimari sayesinde doğru kod bulunur ve çalıştırılır. Bileşenlerin gerçekleştirebileceği işlevler üzerinde Eclipse'in hiç bir sınırlaması yoktur. Geliştiriciler, başka bileşenleri düşünmeden ve etkilemeden bir bileşen üzerinde geliştirme yapabilmektedirler.

Eclipse bu mimari yapıyı sağlamak için, OSGi R4 tanımı [2] üzerine geliştirdiği bileşen mimarisini kullanmaktadır.

2.1. OSGi

OSGi platformu, Java üzerinde çalışmakta ve birbiri ile etkileşebilen bileşenlerin bir araya gelerek ihtiyaç duyulan uygulamayı oluşturması için gerekli olan servisleri içermektedir. Bileşenler, çalışan bir OSGi platformuna yüklenebilmekte ve dinamik servis bulma özellikleri sayesinde uygulamaların yeniden başlatılmasına gerek kalmadan sağladıkları servisler kullanılabilmektedir.

OSGi platformu, bileşenlerin sağladıkları servislerin web servislerinde olduğu gibi sorgulanmasını sağlamaktadır. Ancak, web servisleri kullanımı için mutlaka bir haberleşme katmanı ve çeşitli protokollerin kullanılması gerekirken, OSGi platformunda aynı işlev metot çağrımları ile yapılmaktadır. Bu farklılık, direkt metot çağırımları ile çalışan OSGi'nin, bir web servis çağırımından çok daha hızlı olmasını sağlamaktadır. Ayrıca bileşenler, servis takipçileri sayesinde OSGi platformuna kayıtlanan ve platformdan ayrılan servisleri takip edebilmektedirler.

Bu üstünlüklerin yanı sıra, OSGi platformu standart Java güvenlik mimarisinin üzerine yaptığı eklentiler sayesinde, bileşenler arasında dinamik olarak kullanım haklarının tanımlanmasına da olanak sağlamaktadır.

OSGi platformunun kullanılması sayesinde, bu platforma uygun olarak geliştirilen ve daha önceden testleri yapılmış hazır bileşenlerin tekrar kullanımı sağlanmaktadır. Bileşenlerin entegrasyonu kolaylaşmakta, böylelikle ürünün ortaya çıkma süresi ve maliyeti düşmektedir.

Ayrıca, OSGi platformunda tanımlanmış, kullanıma hazır eklentiler ve servisler bulunmaktadır. Bunlar istendiği zaman platforma takılarak kullanılabilmektedirler. Bunlardan bir kaçı şöyle sıralanabilir; HTTP sunucuları, güvenlik servisleri, XML çözücüleri, kullanıcı ve bileşen yapılandırma servisleri, günlük tutma servisleri v.b.

2.2. Eclipse Zengin İstemci Platformu

Eclipse Zengin İstemci Platformu; üzerine uygulama geliştirebilecek bir tabanı olan, bir bileşen çerçevesi sunan, kullanılabilecek görsel bileşenleri ve bunları kullanarak hazırlanmış görsel bir çalışma alanı bulunan bir platformdur.

Bu platformun üzerine uygulamaya yönelik bileşenler eklenerek özelleşmiş uygulamalar geliştirilebilmektedir. Böylece sıfırdan bir uygulama tasarlamak yerine, gerekli bileşenleri ekleyerek ve platformun sağladığı test edilmiş ve kanıtlanmış özellikleri kullanarak hızlı bir uygulama geliştirmek mümkün olmaktadır. Ayrıca, hazır veya daha önceden geliştirilmiş bileşenlerin tekrar kullanımı ve hızlı entegrasyonu sağlanabilmektedir.

Eclipse Zengin İstemci Platformu [3], yukarıda anlatılan özellikleri sağlamak için gerekli olan, en ufak Eclipse bileşenleri kümesidir. Eclipse'de tanımlanmış genişleme noktalarını gerçekleyen bileşenler yardımıyla grafiksel olan veya olmayan her türlü uygulama, Eclipse tabanlı olarak gerçekleştirilebilmekte ve Eclipse'in sağladığı özelliklerden yararlanabilmektedir. Bu özelliklere, Eclipse güncelleme yöneticisini kullanarak yazılım güncellemesinin yapılması, aranabilir ve zengin içerikli yardım sayfalarının hazırlanması ve gösterilmesi, bileşen yöneticisi vb. örnek olarak verilebilir.

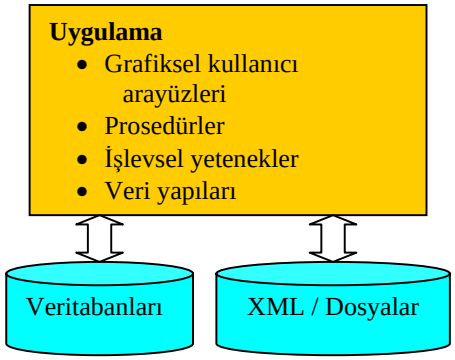
Eclipse Geliştirme Ortamı da aslında Eclipse Zengin İstemci Platformu üzerine geliştirilmiş bir geliştirme ortamı uygulamasıdır. Bu uygulamada örnekleri görülen görünümler, düzenleyiciler, sihirbazlar, yapılandırma öğeleri, perspektif kullanımı, standart ağaç ve tablo grafiksel öğeleri vb. birçok özellik Eclipse Zengin İstemci Platformu'nun kendi içerisinde bulunmakta ya da platforma eklenerek kullanılabilmektedir. Bileşen geliştiricileri; bu eklentileri ve içerdikleri öğeleri, tanımlı genişleme noktalarını kullanarak istedikleri gibi özelliştirebilmektedirler. Hatta istenirse tamamen farklı grafiksel öğeler içeren bileşenler kullanılarak, çok farklı görünümlere sahip uygulamalar geliştirilebilmektedir.

Şu anda pek çok ticari ya da açık kaynak kodlu uygulama, Eclipse Zengin İstemci Platform'u üzerine geliştirilmektedir.

3. Geleneksel Mimariler

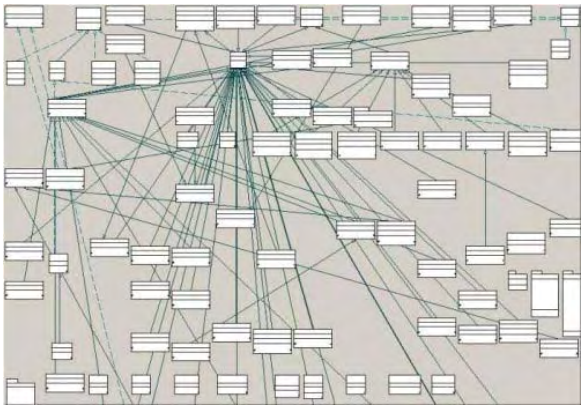
Geleneksel mimarilerde bir yazılım tek parçadan oluşmaktadır [4]. Yazılımın veri yapıları, kullanıcı arayüzleri, işleyiş mantığı ve işlevleri bir arada

bulunmaktadır. Katmanlı bir yapı yoktur. Böyle bir yapı Şekil 2'de verilmiştir.



Şekil 2 Geleneksel yazılım mimarisi

Bu tip bir mimariyi, nesneye dayalı bir şekilde gerçeklediğimizde ortaya
Şekil 3'deki gibi birçok nesne ve nesneler arası ilişki çıkacaktır.



Şekil 3 Nesneler ve ilişkileri

Bu mimarinin dezavantajları şöyle sıralanabilir:

- Yazılımın işlevleri, başka bir yazılımda tekrar kullanılmak istendiğinde, birçok nesne ve ilişkiyi bu mimariden çıkarıp yeni yazılımın mimarisine oturtmak gerekmektedir. Bu yeniden yapılandırma zor ve zaman kaybettiricidir.
- Yazılımda yapılacak hata düzeltmeleri ve yeni eklenecek özellikler bütün yazılımı etkileyecektir. Bu etki yüzünden, yapılan her değişiklikten sonra yazılımın bütün özelliklerinin en baştan test edilmesi gerekmektedir.
- Kullanıcı arayüzü yazılımla iç içe olduğundan başka platformlar için kullanıcı arayüzü geliştirilmesi çok zor olacaktır.

Tek parça halinde yapılan yazılımlarda görülen bu sıkıntılar nedeniyle katmanlı bir yapıya (sunum, iş ve veri katmanları) geçilmiştir. Fakat bu katmanlı yapıda dahi sıkıntılar yaşanması ile bileşen tabanlı mimariler ortaya çıkmıştır.

4. Bileşen Tabanlı Mimariler ve Eclipse

Bir yazılım bileşeni; önceden tanımlanmış işlevleri ve servisleri bünyesinde barındıran, bu servisleri başka bileşenlerin de kullanabilmesi için gerekli servis arayüzü tanımlarına ve haberleşme altyapılarına sahip olan bir yazılım parçasıdır. Bir yazılım bileşeni; tekrar kullanılabilir olmalı, başka bileşenlerle entegre edilebilmeli ve kara kutu şeklinde çalışmalıdır. Yani bileşen, iç tasarımı ve detayları bilinmeden sadece tanımlanmış servis arayüzleri aracılığıyla kullanılabilmelidir. Her yazılım bileşeni bir geliştirme birimidir ve uygulama yazılımının bütününe ihtiyaç duymadan geliştirilebilmelidir. Böylece bileşen kendi başına test edilebilir olmaktadır.

Bileşen tabanlı mimariler, birbirinden bağımsız geliştirilen yazılım bileşenlerinin birbirleri ile entegre edilerek bir uygulama yazılımı oluşturulabileceği düşüncesiyle ortaya çıkmıştır. Nasıl mekanikte ya da elektronikte arayüzleri tanımlı bileşenler birleştirilip bir ürün ortaya çıkarılabiliyorsa, yazılım bileşenleri de birleştirip bir uygulama yazılımı oluşturabilmektedir. Böylece, test edilmiş ve güvenilir bileşenler kullanılarak hızlı bir şekilde uygulama yazılımı geliştirilebilir ve sadece bileşenlerin arasındaki entegrasyon test edilerek ürün ortaya çıkarılabilir.

Böyle bir mimarinin uygulandığı en güzel örneklerden biri Eclipse Geliştirme Ortamı'dır. İçerdiği 10 alt proje kapsamında birbirinden bağımsız şekilde geliştirilen bileşenler sayesinde, şu anda dünyada Java dili için en çok kullanılan geliştirme ortamı [5] olmuştur. Ayrıca bu projelerin dışında birçok açık kaynak kodlu ya da ticari bileşen Eclipse ekosistemine eklenerek diğer bileşenlerle yan yana kullanılabilmektedir.

Eclipse'in bu başarısı ve bileşen tabanlı mimarisinin sağladığı yararlar IBM, Borland, WindRiver gibi birçok büyük firma tarafından görülmüş ve bu firmalar, yazılımlarını Eclipse Zengin İstemci Platformu üzerine geliştirmeye başlayarak, bu mimarinin faydalarından yararlanma yoluna gitmişlerdir [6].

Örneğin, Borland firması yıllardır Java geliştirme ortamı olarak geliştirdiği Borland Jbuilder ürününü, 12. sürümünden sonra bırakarak Eclipse tabanlı başka ürünler geliştireceğini açıklamıştır [7].

Başka bir örnek ise NASA'nın Mars gezegeni araştırmalarında kullanmak üzere Eclipse Zengin İstemci Platformu üzerine geliştirdiği yazılımlardır [8].

5. Komuta Kontrol Sistemleri

Komuta kontrol sistemleri, askeri bir operasyonun bütün safhaları ve unsurları ile komuta ve kontrol edilmesini sağlayan sistemlerdir. "Sistemler sistemi" olarak da tanımlanan bu sistemler, üstlendikleri görevlere özel birçok farklı ve karmaşık işlevi yerine getirebildikleri gibi görevden bağımsız olan birçok ortak işlevi de barındırmaktadırlar.

Ortak işlevlere örnek olarak:

- Haberleşme ve mesaj yönetimi
- Güvenlik ve kullanıcı yapılandırması
- Yazıcı işlemleri
- Sistem yedekleme
- Sistem imha

vb. sayılabilir. Birçok komuta kontrol sisteminde ayrıca coğrafi bilgi sistemi yetenekleri de bulunmaktadır. Göreve yönelik işlevlere örnek olarak da:

- Ateş destek işlemleri
- Meteoroloji işlemleri
- Yer ölçme işlemleri
- Gözetleme işlemleri

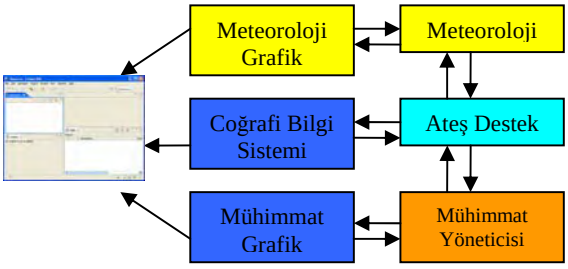
vb. sayılabilir. Bu ortak ve göreve özel işlevlerin farklı şekillerde bir araya gelmesi ile farklı komuta kontrol sistemleri ortaya çıkmaktadır.

Bu durum, bileşen tabanlı mimarilerin komuta kontrol yazılımlarının geliştirilmesinde kullanılması için çok uygun olduğunu ortaya koymaktadır. Komuta kontrol yazılımlarının işlevlerini yerine getirecek bileşenler geliştirilip, daha sonra birbirleri ile hızlı bir şekilde entegre edilerek; güvenilir ve işlevsel komuta kontrol yazılımları ortaya çıkarılabilecektir.

Bu bileşenler, Eclipse Zengin İstemci Platformu'na uygun şekilde geliştirilerek kendi grafiksel öğelerini komuta kontrol yazılımına eklemeleri sağlanabilir ya da özelleşmiş arayüzler hazırlanarak bu bileşenlerin servisleri kullanılabilir. Her iki durumda da bileşenlerin sağladıkları servisler kullanılacağı için güvenilir bir işlevsellik sağlanmış olacaktır.

Örnek bir komuta kontrol yazılımı bileşenlerinden bazıları, bunların kendi aralarındaki ve Eclipse Zengin İstemci Platformu ile arasındaki ilişkiler Şekil 4'te gösterilmektedir. Örnekte meteoroloji işlevini gerçekleştiren bir bileşen ile bu bileşeni kullanan grafiksel bir arayüz, ilgili menü ve araç çubuğu düğmelerini içeren diğer bir bileşen bulunmaktadır.

Platformda bulunan bileşenlerden bazıları, meteoroloji bileşeninin tanımlı servisini kullanarak kendilerinde bulunan meteoroloji bilgilerini Meteoroloji bileşenine aktarmaktadırlar. Bu bilgileri değerlendiren meteoroloji bileşeni de yine tanımlı başka bir servisi ile geçerli meteoroloji raporunu, isteyen diğer bileşenlere sağlamaktadır. Örnekte bulunan Ateş Destek bileşeni, ateş destek görevilerini yerine getirmek için ihtiyaç duyduğu meteoroloji ve mühimmat bilgilerini ilgili bileşenlerin tanımlı servislerini kullanarak elde etmektedir. Ayrıca seçili hedef koordinatlarını da Coğrafi Bilgi Sistemi'nin servislerini kullanarak almaktadır. Bu işlemlerin ardından görev yaratılmakta ve Coğrafi Bilgi Sistemi'nde güncellemeler yapılmaktadır.



Şekil 4 Örnek Mimari

Ayrıca sol kolonda gösterilen bileşenler sağ kolondaki işlevsel bileşenler için grafiksel arayüzler sağlamaktadır. Eclipse Zengin İstemci Platformu'na kendi ekranlarını, düzenleyicilerini, yardım ekranlarını, menülerini, araç çubuklarını vb. grafiksel öğelerini ekleyerek grafiksel bir komuta kontrol uygulaması meydana getirmektedirler.

İşlevsel bileşenler ile grafiksel kullanıcı arayüzü bileşenlerinin ayrımının yapılması sayesinde, işlevsel bir bileşene (ör. Meteoroloji bileşeni) farklı grafiksel kullanıcı arayüzlerinin kazandırılması mümkün olmaktadır. Farklı bir grafiksel kullanıcı arayüzü istendiğinde, bu değişiklik Grafiksel Kullanıcı Arayüzü bileşenin değiştirilmesi ile kolaylıkla yapılabilecektir. Hatta OSGi platformunun verdiği dinamiklikle, bu değişiklik çalışan bir sisteme uzaktan yükleme yapılarak ve uygulamanın yeniden başlatılmasına gerek olmadan sağlanabilecektir.

Aynı şekilde veri katmanının da farklı bileşenlerde gerçeklenmesi, uygulamaya daha fazla esneklik sağlayacaktır. Veri katmanı tek bir bileşende gerçeklenebileceği gibi her bileşen için ayrı bir veri katmanı bileşeni de geliştirilebilecektir.

Örneklerden de anlaşılacağı üzere bir bileşen daha küçük alt bileşenlere bölünüp gerçekleştirilebilmektedir. Bu tip ayrıştırmalar yazılıma esneklik katıyor olsa da çok büyük sistemlerde çok sayıda bileşen olması bileşen yönetiminin, sürümlendirmesinin iyi yapılması gerekliliğini ortaya çıkarmaktadır.

6. Sonuç

Bileşen tabanlı mimarilerin, geleneksel yazılım mimarilerine üstünlükleri; tekrar kullanılabilirliği artırması, geliştirme zamanını azaltması, test edilmiş ve güvenilir bileşenlerin kullanımı sayesinde de güvenilir yazılımlar elde edilmesi olarak sıralanabilir. Ayrıca bileşenler tek başlarına bir geliştirme öğesi olacakları için bileşen geliştirme takımlarının kendi başlarına çalışmalarına imkan vermektedir.

Eclipse platformu ve alt projeleri, OSGi platformunun da yardımı ile yazılım dünyasına yeni bir geliştirme ortamı sağlamanın ötesinde, bileşen tabanlı mimarilerin uygulanması için çok uygun bir ortam sağlamaktadırlar.

Özellikle, Eclipse Zengin İstemci Platformu şu anda birçok açık kaynak kodlu veya ticari uygulamanın alt yapısı olarak kullanılmaktadır. Eclipse Geliştirme Ortamı'nın ve bu ürünlerin başarısı, Eclipse Zengin İstemci Platformu'nun başarısının kanıtıdır.

ASELSAN'da bileşen tabanlı mimarilerin yazılımlara ve geliştirme sürecine olan olumlu etkilerinden yararlanmak için, çeşitli ortak bileşenler geliştirilmektedir. İlk etapta, Haberleşme ve Coğrafi Bilgi Sistemi gibi komuta kontrol sistemlerinin ana bileşenleri geliştirilmektedir. Başka bileşenlerin de geliştirilmesi ve uygulama yazılımları tarafından kullanıma alınması belirlenen plan çerçevesinde devam etmektedir.

7. Kaynakça

[1] Eclipse Foundation, *Eclipse Web Sitesi*, http://www.eclipse.org

[2] The OSGi Alliance, *OSGi Service Platform Core Specification Release 4*, http://www.osgi.org, Ağustos 2005

[3] Eclipse Foundation, *Eclipse Zengin İstemci Platformu Web Sitesi*, http://www.eclipse.org/rcp

[4] Hamid Ben Malek, *Oasis ebSOA An Introduction to Service Oriented Architecture*, http://www.oasis-open.org

[5] Darryl K. Taft, *Eclipse Dominates Grassroots IDE Survey, Sets Tone for Open-Source Java*, http://www.eweek.com/article2/0,1895,2017113,00.asp, 18 Ekim 2006

[6] Darryl K. Taft, *Eclipse RCP Adoption Nearly Triples*, http://www.eweek.com/article2/0,1895,2011851,00.asp, 5 Ekim 2006

[7] Darryl K. Taft, *Borland to Divest Dev Tools with Segue Buyout*, http://www.eweek.com/article2/0,1895,1922024,00.asp, 8 Şubat 2006

[8] Eclipse Foundation, *NASA Uses Eclipse for Interplanetary Operations*, http://www.eclipse.org/community/casestudies/NASAfinal.pdf

Uygulama Mimarisi ve Altyapı - II

Servis Odaklı Mimari Prensipleri ve ROTA Uygulama Çatısı

Serkan Üstündağ
Bimar, Yazılım Uzmanı
serkan.ustundag@izm.bimar.com.tr

1. Giriş

Son yıllarda; nesneye yönelik dillerin kullanımının yaygınlaşması ve dağıtık programlama uygulamalarının XML ve web servisleri teknolojileri ile kolaylaşması sonucunda, fonksiyonların servisler halinde sunulmasını benimseyen servis odaklı mimari, uygulama mimarileri arasında hızla önem kazanmaya başlamıştır.

Özellikle kurumsal uygulamalar arasında varolan entegrasyon çözümlerinin maliyetleri, uygulamaların servis odaklı mimari prensiplerine göre geliştirilmesi durumunda önemli bir oranda azalmaktadır ve entegrasyonların kalitesi ve güvenilirliği artmaktadır. Kurumlar, servis odaklı mimari geliştirmeyi bilgi işlem stratejisi olarak belirlemeye başlamıştır ve bu doğrultuda çalışanlarını eğitmekte ve yönlendirmektedir.

Öncelikle, Rota uygulama çatısı servis odaklı mimari prensiplerine uygun olarak geliştirildiği için servis odaklı mimarinin temel prensipleri anlatılacak, servis tiplerinden bahsedilecektir.

Bir sonraki bölümde ise iş süreçlerinin yönetimi ve servis odaklı mimari arasındaki ilişki anlatılmaktadır. Takip eden kısımda süreç bütünlüğü kavramı ile veri bütünlüğü kavramı arasındaki fark tanımlanmakta, süreç bütünlüğünün sağlanması için gerekli olan hareketlerin tipleri ve özellikleri hakkında bilgi verilmektedir.

Son yıllarda servis odaklı mimarilerin önemli bir parçası haline gelen web servisleri kullanıldığı taktirde, süreç bütünlüğünü ve yönetimini sağlamak için tanımlanmış olan standartlardan, Rota uygulama çatısı hakkından bilgi veren bölümden önce kısaca bahsedilmektedir.

Son kısımda, Rota uygulama çatısının özellikleri üzerinde durulacak, sağladığı faydalardan bahsedilecektir.

2. Servis Odaklı Mimari Nedir?

Servis Odaklı Mimari bir bilgi işlem stratejisi olup, bu stratejinin amacı kurumsal yazılımlar içindeki farklı fonksiyonları karşılıklı çalışabilecek, standartlara dayanan, iş ihtiyaçlarını karşılamak üzere kolaylıkla tekrar kullanılabilen ve birleştirilebilen servisler haline sokmaktır.

Servisler, aslında mevcutta var olan fonksiyonların yada yeni oluşturulan fonksiyonların, belli prensipler göz önünde bulundurularak servis halinde sunulmasıdır. Bu kısa sürede gerçekleştirilebilecek bir hedef olamayacağından bir kurum ancak bunu stratejik olarak tasarlayıp, bu stratejiye ulaştıracak olan yol haritasını takip ederek zamanla hayata geçirebilecektir.

3. Servis Odaklı Mimarinin Genel Özellikleri

Yeniden Kullanılabilir Olmalıdır

Servisler Ortak Bir Kontratı Paylaşmaktadır

Servis kontratları aşağıdaki tanımlamaları standart bir şekilde yapmaktadır;
-Servis uç noktası,
-Servis operasyonları,
-Her operasyon tarafından desteklenen girdi çıktı mesajları,
-Servisin ve operasyonun kuralları ve özellikleri

Kontratlar genellikle servis odaklı mimarinin temel kısımlarını tanımlamaktadır. İyi bir servis kontratı aynı zamanda servisin belirli bir görevi nasıl yaptığını açıklayan mantıksal bilgi de sağlamalıdır.

Servisler Gevşek Bağlıdır

İstekler sürekli değiştiğinden dolayı uygulama ortamları da sürekli değişmektedir. Bu değişimlerin önceden planlanmasına imkan yoktur. Servisler bağımlılıkları, servis kontratları kullanarak azaltmaktadırlar.

Servisler Çalışma Mantığını Gizlemektedir

Servisler uygulama detaylarını dış dünyadan saklayarak kara kutu gibi davranmaktadırlar. Servis isterse uygulama mantığını iki ayrı sisteme açabilir.

Servisler Birleştirilebilir

Servisler başka kaynaklardan gelen iş mantığını simgeleyebilmektedirler. Buna diğer servisler de dahildir. Bu prensibin temel amacı, ihtiyaç duyulduğu zaman servislerin, başka bir servisin kompozisyonunda katılımcı olarak yer almasını sağlamaktır. Birleştirilebilirliği vurgulayan temel servis odaklı mimari ilavesi orkestrasyondur.

Kompozisyon, hangi uygulama servisi kullanacak bilmeksizin, servisin fonksiyonelliğinin yeniden kullanımı ve modülerliğine odaklandığından dolayı etkili bir tasarımdır .

Servisler Bağımsız Çalışabilmelidir

Bir servis başka bir servisten bağımsız ve etkilenmeden çalışabilmelidir. Yani bir servisi kullanabilmek için başka bir servisin orada hazır olmasına gerek olmamalıdır.

Servisler durumsuzdur

Durum bilgisi güncel uygulamada veriye özeldir. Servis bir mesajı işlerken geçici olarak durum içerir. Eğer uzun süre duruma sahip olması gerekirse diğer servislerin isteklerine cevap vermesine engel olunacaktır.

Servisler keşfedilebilir

Servisler çalışma zamanında dinamik olarak keşfedilmelidir. Servisin müşterisi kayıtçıyı tarayarak servisi çalıştırmak için gerekli olan tüm bilgileri bulmaktadır. Servise bağlanmak için derleme bağımlılığı yoktur .

4. Servis Tipleri

Uygulama önuçları

Servis olmamalarına rağmen SOA' nın aktif elemanlarındandır. Tüm iş süreçlerini başlatır ve sonuçlarını alırlar. Grafiksel arayüzler ve küme ("batch") işlemler uygulama önucu örnekleridir .

Temel Servisler

Servis odaklı mimarinin temelidir. Veri odaklı yada mantık odaklı olarak ikiye ayrılmaktadırlar. Veri odaklı servisler verinin saklanması, verinin elde edilmesi, kilitleme mekanizması ve hareket yönetimi gibi kavramları içermektedirler. Mantık odaklı servisler ise içerisinde karmaşık iş kuralları ve kompleks algoritmalar barındırmaktadır. Birçok servis hem veri hem de iş mantığı ile ilgili olduğundan kesin bir ayrım yapılamamaktadır .

Aracı Servisler

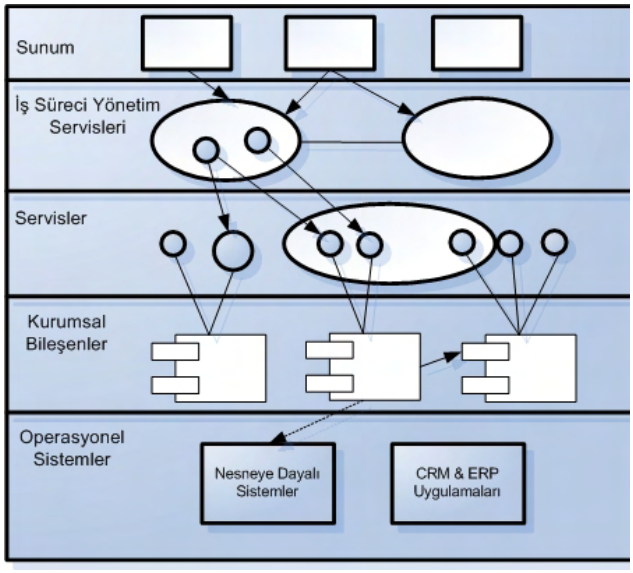
Mimarideki teknik uyumsuzlukları ve tasarım ayrıklıklarını köprü görevi görüp çözen durumsuz servislerdir. Servis odaklı mimaride hem istemci hem de sunucu konumundadırlar .

Süreç Merkezli Servisler

Organizasyonun iş süreci bilgisini içermektedirler. Servis odaklı mimaride hem istemci hem de sunucu görevindedirler ve sürecin durumunu yönetmektedirler. Etkin bir gerçekleştirim için dikkatli bir tasarım gerektiren servislerdir. Aracı servislerden temel farkı, istemciler için sürecin durumunu yönetmek zorunda olduğundan durum bilgili olmasıdır .

Genel Kurumsal Servisler

Şu ana kadar anlatılan servisler organizasyon sınırları içerisinde yer alan servislerdir. Genel servisler ise ortaklar ve müşteriler ile iletişim halinde olan servislerdir. Örneğin bir nakliye şirketi müşterilerinin yüklerini takip edebilmesi için bir servis sunabilir .



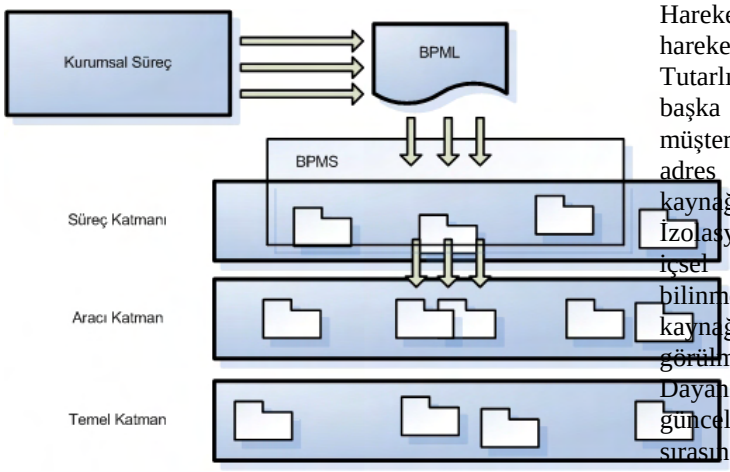
Şekil 1. Servis Odaklı Mimari Katmanları

5. Servis Odaklı Mimari ve İş Süreçleri Yönetimi

Bilişim tarafında BPM (Business Process Management) denildiğinde süreç modelleme ve iş akışı yönetimi öne çıkmaktadır .

BPMS (Business Process Management System), BPM yönetimini sağlayan teknik ortam sağlamaktadır. BPM motoru, tasarım araçları, iş süreçleri izleme ve modelleme yetenekleri gibi parçalardan oluşmaktadır .

İş süreçleri ile ilgili önemli kuralları ve bilgileri uygulama kodu yerine, uygulama sisteminin dışında, BPM sisteminin kontrolü altına almak en güçlü özelliklerinden birisidir. SOA, BPMS tarafından ihtiyaç duyulan altyapıyı sağlamaktadır .



Şekil 2. SOA ve BPM yaklaşımlarının kesişimi

6. Süreç Bütünlüğü Yönetimi

Birçok alt sistemlerden oluşan karmaşık iş süreçlerinin tutarlılığını sağlamak bilişim sektöründeki en uğraştırıcı problemlerden birisidir.

Süreç bütünlüğü çok iyi tanımlanmış bir kavram değildir. Süreç bütünlüğünü oluşturan çekirdek yapılar veri bütünlüğü kavramlarının kurulumuna bağlıdır. Ancak veri bütünlüğü, karmaşık iş süreçlerinin bütünlüğü sağlamak için ihtiyaç duyduklarını karşılamakta yetersizdir. Bu yüzden süreç bütünlüğü kavramı oluşturulmuştur .

Süreç bütünlüğünü sağlamak için birçok çözüm bulunmaktadır. Bu çözümler arasında günlükleme ve izleme gibi basit teknik çözümlerin yanında ileri seviyede hareket kavramları bulunmaktadır. BPM sistemleri teknik çözümlerden ziyade daha çok iş odaklı çözümler sunmaktadır .

ACID Hareketler

OLTP sistemleri paylaşılan verinin birçok kullanıcı tarafından eş zamanlı olarak işlenmesine olanak sağlamaktadır. Hareket kelimesi, OLTP sisteminde verinin durumunu değiştiren birim iş olarak tanımlanmaktadır. En üst seviyede veri bütünlüğünü taahhüt eden, eş zamanlı ve dağıtık çalışabilen hareketlerin ideal karakteristik özelliklerini belirmek için ACID kavramı ortaya çıkarılmıştır .

ACID Nedir?

ACID, hareketlerin karşılaması gereken şu dört ihtiyacı tanımlamak için kullanılan bir yoldur:

Bölünmezlik (“Atomicity”): Hepsi yada hiçbiri prensibine dayanan atomik (“atomic”) iş birimleridir. Hareket sırasında bir hata oluşması durumunda tüm hareket geri alınmaktadır.

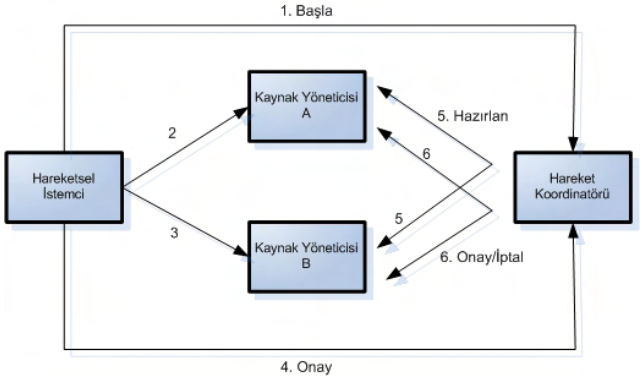
Tutarlılık (“Consistency”): Veriyi tutarlı bir durumdan başka bir tutarlı duruma çevirmelidir. Örneğin bir müşteri hesabının silinmesi durumunda müşteriye ait adres bilgileri de silinmelidir . Bir hareket, veri kaynağının tutarlı bir durumda kalmasını sağlamalıdır.

İzolasyon (“Isolation”): Çalışmakta olan bir hareketin içsel durumu bir diğer hareket tarafından bilinmemelidir . Tamamlanmamış hareketler veri kaynağının diğer kullanıcıları tarafından görülmemelidir.

Dayanıklılık (“Durability”): Hareketin onaylanmış güncellemeleri süreklidir. Hareketin çalışması sırasında hata oluşması durumunda sistem hareket başlamadan önceki haline geri döndürülür . Bir hareket veritabanına yazıldıktan sonra kalıcı ya da dayanıklı olmalıdır.

Hareket İzleme ve Dağıtık İki Aşamalı Onay (2PC)

Farklı sistemlerin etkileştiği bir uygulamayı, ACID özellikleri taşıyacak şekilde gerçekleştirilmek zordur. Şekil 3’ te görüldüğü gibi, birden fazla veri tabanı veya hareketsel kaynak üzerinde çalışan bir hareketin ACID özelliklerde olmasını sağlamak için Hareket Koordinatörü kullanılabilir .



Şekil 3. Hareket Koordinatörü

Farklı kaynaklardaki hareketler, hareket izlemenin bir parçası olan hareket koordinatörü tarafından düzenlenmektedir. Her dağıtık hareketin sonunda koordinatör katılımcı kaynak yöneticilerindeki hareketin onaylanmasını ayarlamaktadır :

- İlk aşamada (hazırlanma) tüm katılımcı kaynak yöneticileri, onay veya ret cevabını göndermek için hazırlanırlar.
- İlk aşamanın sonuçlarına göre koordinatör katılımcıları onayların yada işlemi geri alın diye bilgilendirir.
- Tek bir iptal oyu bile tüm hareketin geri alınmasına yeterlidir. Tüm katılımcıların onay vermesi durumunda tüm değişiklikler kalıcı hale getirilmektedir.

ACID hareketler tek bir veritabanı yada dağıtık sistemler için bütünlüğü sağlamak açısından iyi bir teorik kavram olmasına rağmen, gerçek uygulamalarda kullanılması pratik olmayan yapılardır. .

Hareketsel Adımlar

Hareketsel adım, tek bir hareket kapsamında işletilen birbirleri ile alakalı aktiviteler kümesidir. Hareketsel adımlar, karmaşık ve uzun süreli iş süreçlerinin, hareketsel bütünlük sağlayan ve kısa süren özel adımlara ayrılmasına yardım ettiğinden, süreç bütünlüğü için anahtar kavramdır. Ayrıca dağıtık sistemlerin dayanıklılığını ve esnekliğini arttırmaktadır.

Hareket Zincirleri ve Telafi

Özel adımların birbirine bağlanması ile karmaşık iş akışları ve süreçler yaratılabilmektedir. Hataların tespit edildikten sonra düzeltme işlemlerinin de yapılması gerekmektedir .

Olası bir çözüm, hata oluşması durumunda telafi hareketinin işletilmesi, mantıksal olarak bir önceki hareketin geri alınmasıdır. Örneğin hesaptan para düşmenin telafi hareketi hesaba düşen para kadar ilave yapmaktır .

Dikkat edilirse hareket zincirleri, ACID özellikli hareketlerin yalıtım özelliği konusunda daha esnek bir davranış sergilemektedir. Zincirin her bağlantısının sonucu dış dünya tarafından erişilebilir durumdadır.

Bir iş akışında telafi hareketlerinin uygulanması için, iş akışının adımlarının girdileri veya çıktılarının günlüklenmesi gerekmektedir. Bu veriler telafi hareketleri için girdi oluşturabilecektir .

Özetle, ACID hareketler karmaşık iş akışları için yetersizdir. Bu tip hareketler yerine, telafi hareketleri içeren hareket zincirleri, süreç bütünlüğü sağlanması için daha iyi bir yöntem sunmaktadır. Hareket zincirleri, özel hareket adımlarını birleştirip karmaşık iş akışlarına koymaktadır. Telafi hareketleri, zincirde yer alan bir hareketin çalışması sırasında bir problem oluşması durumunda, önceden çalıştırılmış olan adımların yaptıklarını geri almaktadır .

7. Web Servisleri ve İş Süreçleri

Web servisleri uygulamaların interaktif olarak haberleşebilmesi ve çoklu işlemlerin otomatik ve ortaklaşa çalışabilmesi için yeni olanaklara imkan vermektedir. Bunun ötesinde, Web servis standartları ile mevcut dağıtımlı sistemler üzerinde majör değişiklikler yapmadan, bütünüyle yeniden kullanılabilir ve yaygın olarak erişilebilir yazılım uygulamaları geliştirilebilmektedir .

Büyük ölçekli sistemlerde işlemlerin dağıtımlı olarak gerçekleştirilebilmesi için mevcut sistemlere iş akışı ve iş süreç yönetim çözümleri de ilave edilmektedir. Bu tür uygulamalar senkron/asenkron haberleşme desteğine, uzun çalışabilen işlemlere, karma servisleri kullanma olanağına, durum, akış ve hata kontrollerine ihtiyaç duymaktadır. “Web servis orkestrasyon” konusu tüm bu olayları kapsamaktadır .

Web Servis Orkestrasyon Standartları

Web servis orkestrasyon standartları, Web servis yığın yapısını iş süreç yönetim özellikleri ile genişletmektedir. Önerilen standartlar genel olarak interaktif çalışan ortak Web servisleri birbiri ile ilişkilendirme imkanı sunar .

OASIS grubu “Business Transaction Protocol (BTP)” standardını sunmuştur. IBM’ in hazırladığı “Web Service Flow Language (WSFL)” ve Microsoft’un hazırladığı “XML Language (XLANG)” birleşerek IBM, Microsoft ve BEA tarafından desteklenen “Business Process Execution Language for Web Services (BPEL4WS)” standardına

dönüşmüştür. BPEL4WS'in tamamlayıcı iki protokolü mevcuttur; mesaj koordinasyonu sağlayan "WS-Coordination" ve işlem hareketlerine yönelik hazırlanan "WS-Transaction". BPMI.org grubu "Business Process Modeling Language (BPML)" isimli bir çalışma sunmuştur. SUN ise "Web Services Choreography Interface (WSCI)" standardını desteklemektedir.

WS-Coordination

WS-Coordination tarifnamesi, aktivitelerin koordine edilebilmesi için bir koordinatör ile koordinasyon protokollerini kullanan genişletilebilir uygulama çatısı tanımlamaktadır. Uygulama çatısı, dağıtık aktivitelerin sonuçlarının katılımcılar tarafından tutarlı bir şekilde alınmasına olanak sunmaktadır. Uygulama çatısında tanımlanan koordinasyon protokolleri, hem basit kısa süreli operasyonlar hem de karmaşık uzun süreli iş aktiviteleri için protokollere yer vermektedir .

Her koordinasyon servisi, aktivitenin çalışma mantığını belirleyen bir koordinasyon tipine bağlıdır. Koordinasyon tipleri farklı tarifnamelerle özelleştirilmişlerdir. Ancak ilk olarak düşünülen koordinasyon tipleri WS-AtomicTransaction ve WS-BusinessActivity dir. Bu tarifnamelere özel kavramlar WS-AtomicTransaction ve WS-BusinessActivity kısmında anlatılmaktadır .

WS-AtomicTransaction

WS-AtomicTransaction tipi ACID hareketler gibi çalışmaktadır. Aktivitenin tümü başarılı olduğu zaman onaylayan, aksi durumda her şeyi ilk durumuna döndüren mekanizmayı gerçekleştirmektedir .

WS-BusinessActivity ile Uzun Süreli Hareketler

İş hareketlerinin genellikle birkaç atomik hareket içermektedirler. Bir atomik hareket tarafından güncellenmiş olan kaynaklar tüm iş hareketinin başarısız olması durumunda ilk hallerine geri dönmelidirler. Telafi bu sorunun çözümü için yaygın kullanılan çözüm metodolojilerinden birisidir..

Atomik hareket çalışması süresince aktivitenin kullandığı kaynaklar işlem bitene kadar genellikle kilitlenmektedir. Bu tip işlemlerin yaşam süresi kısa olduğundan dolayı sistem kaynaklarının kullanımı açısından bir sorun çıkmamaktadır .

WS-BusinessActivity de WS-AtomicTransaction gibi WS-Coordination uygulama çatısının özelleştirilmiş hali olan bir koordinasyon tipidir. Birbirine çok benzeyen iki protokol sunmaktadır. Bu protokoller katılımcının aktivite sırasında nasıl davranması gerektiğini belirtmektedirler.

BPEL4WS

Business Process Execution Language for Web Services (BPEL4WS) XML tabanlı bir programlama dilidir. Bu standart ile Web servis etkileşimi tanımlanabilir ve birlikte çalışma (cooperation) protokolleri kullanılabilir. Ayrıca BPEL4WS dili, iş süreçleri oluşturmak amacıyla farklı aktivitelerin birleştirilmesinde de kullanılabilir.

BPEL4WS web servisleri için geliştirilmiş olan iş akışı tabanlı birleşim dilidir. Farklı tiplerde basit aktiviteler içermektedir. Bunlar sayesinde uygulamaların birbirlerini tetiklemelerine ve mesajlaşabilmelerine olanak sunmaktadır. Etkileşim sırasında bekleme sağlamakta yada veriyi bir noktadan diğer bir noktaya taşımayı desteklemektedir. Hata durumlarını gösterebilmekte ve tüm bileşimi sonlandırabilmektedir .

Bu basit aktiviteler yapısal aktiviteler yardımı ile daha karmaşık algoritmalar oluşturma için kullanılabilmektedir. Bunlara örnek olarak belli bir sırada çalıştırma veya koşula göre işletme veya döngüsel işletme gösterilebilir .

BPEL4WS iş akışı modeli aktiviteleri kavrayıp hata ve telafi yakalayıcılarını bu kapsamlar için belirleme yeteneğine sahiptir. Hata yakalayıcı bir istisna fırlatıldığı zaman çalıştırılmaktadır, telafi yakalayıcıları ise hatalara veya telafi aktivitelerine bağlı olarak tetiklenmektedir .

8. Rota Uygulama Çatısı

Yeni Nesil Acentelik (YNA), Arkas Holding'e bağlı acentelik şirketlerinin yazılım gereksinimlerini karşılamak için geliştirilmekte olan bir yazılım paketidir.

Bu paket içerisinde farklı işlevleri gerçekleştiren ve birbiri ile iletişim halinde olan büyük modüller bulunmaktadır. YNA sisteminin alt sistemleri tamamlandıkça kullanıma açılacaktır. Bu yüzden tamamlanan alt sistemlerin şu anda kullanılmakta olan AS400 ve SAP sistemleri ile ortak olarak çalışması gerekmektedir.

YNA yazılımının karşılaması beklenen en temel gereksinimler şunlardır;

- Varolan kurumsal sistemler ile entegre olarak çalışabilmesi
- Müşterilerin sistemleri ile veri alışverişinde bulunabilmesi
- Kullanıcıların ve müşterilerin farklı arayüzlerden servis alabilmesi
- Değişikliklere kolay uyum sağlayabilen bir sistem olması
- Bakım maliyetinin çok yüksek olmaması

Tüm bu gereksinimlerin incelenmesi sonucunda genel prensipleri ve servis kavramı anlatılmış olan servis odaklı mimari yaklaşımının en uygun yaklaşım olduğuna karar verilmiş ve bu doğrultuda çalışmaya başlanmıştır. Bu prensiplerin desteklendiği bir uygulama çatısına ihtiyaç duyulmuştur ve Rota uygulama çatısı geliştirilmiştir.

Farklı sistemlerle entegre bir şekilde çalışması gerekliliği iş kuralı değişikliklerinde sistem bütününün en az şekilde etkilenmesi ihtiyacı, servis odaklı mimari prensipleri incelendiği zaman bu kararda büyük etken oluşturmuştur.

Bu gereksinimler ve mimari karar doğrultusunda Rota adı verilen, hem iş katmanı hem de ön yüz katmanı ile ilgili önyüz, haberleşme ve iş nesneleri ile ilgili temel sınıfları barındıran uygulama çatısı geliştirilmiştir.

Rota uygulama çatısı katmanlı mimari prensiplerine uygun olarak önyüz, iş katmanı ve veri erişim ile ilgili temel sınıfları barındıran, katmanlar arasındaki iletişimin mesaj sınıfları ile sağlandığı, servis prensiplerine uygun olarak iş metotlarının, birbirine bağımlı olmadan, yeniden kullanılabilir ve web servisleri ile dış sistemlere açılabilen servisler olarak geliştirilmesine olanak tanıyan bir uygulama çatısıdır. Şekil 4' te Rota uygulama çatısının genel mimarisi görülmektedir.

İş Katmanı (Servisler ve Kurumsal Bileşenler):

İş katmanında yer alan operasyonlar "BusinessEntity" adı verilen nesnelerde yer almaktadır. Servis olarak nitelendirilebilecek olan işler bu nesnelerde yer alan operasyonları kullanabilmektedir.

Rota uygulama çatısı içerisinde yer alan temel "Transaction" sınıfları kullanılarak yaratılan iş nesneleri, Yna içerisindeki servisleri temsil etmektedir. Daha önceden de belirtildiği gibi bu servisler web servisleri ile dış dünyaya açılmaktadır.

Servisler, işi gerçekleştiren metot çağrımlarında, bir sonraki kısımda anlatılacak olan istek ve cevap nesnelerini almaktadır. İstekteki verileri işleyerek oluşan sonuçları cevap nesnesine aktarmaktadır.

Bu servisler;

- Hangi istemci için iş yaptıklarını bilmezler,
- Bir servis farklı bir servise direk olarak erişemez,
- Servisler dağıtık hareketleri desteklemez,
- Servisin operasyonu içerisinde yer alan bütün işlemler ACID özellikli bir hareket kapsamında yönetilmektedir,

İletişim Katmanı:

Katmanlar arasındaki iletişim istek ve cevap nesneleri aracılığı ile sağlanmaktadır. Önyüzden girilen bilgiler (web arayüzü, mobil cihaz arayüzü veya bir web servisi vb.) istek nesneleri aracılığı ile hizmet verecek olan uygun servise aktarılmakta, servisin işletilmesi sonucunda oluşan bilgiler cevap nesnesi aracılığı ile istekte bulunan varlığa geri dönmektedir.

Rota uygulama çatısında yer alan temel mesaj sınıflarını ata olarak kullanan, uygulama servisine özel mesaj sınıfları üretilmektedir.

Mesajlar;

- Tek bir komut ile XML' e dönüştürülebilmektedir,
- Serileştirilebilir sınıflardır,
- Uygulamaya özel temel sınıf yaratılmasına olanak sunmaktadırlar,
- Başlık ve gövde olmak üzere iki kısımdan oluşmaktadırlar.

Servis Yönetimi ve Yönlendirme Katmanı:

Rota uygulama çatısı, farklı arayüzlerden (grafiksel arayüz, web servisleri vb.) istek mesajı gelmesine olanak sağlaması açısından bir aktarım katmanı içermektedir. Bu aktarım katmanının temel görevleri istek ve cevap nesnelerini yaratmak, çalıştırılması gereken servisi bulmak ve bu servisi tetiklemektir.

Ayrıca bu katman istek ve mesaj sınıfları ile ilgili yapılacak olan ortak işlemleri gerçekleştiren "Pipeline" adı verilen yapıları tetiklemekle görevlidir. "Pipeline" varlıkları günlükleme, yetki kontrolu gibi işlemleri yapan daha küçük yapıları içermektektedir. Bu küçük yapılar farklı amaçlar için oluşturulabilecek olan "Pipeline" larda yeniden kullanılabilmektedirler.

Bu katman ile grafiksel arayüzler arasında bağlantıyı sağlayan, önyüze özel yardımcı bileşenler de Rota uygulama çatısının üyeleridir.

Arayüz Katmanı:

Rota uygulama çatısı farklı arayüzlerden istek gelmesine olanak tanımaktadır. Şu ana kadar geliştirilmiş olan yapıda grafiksel web arayüzü ve web servisi arayüzünden erişime olanak sağlayan bileşenler bulunmaktadır. Bu bileşenler daha önceden de belirtildiği gibi aktarım katmanı ile etkileşime geçerek ilgili servislere ulaşmaktadır.

Örnek Akış Senaryoları

Rota uygulama çatısı kullanılması durumunda kullanıcının arayüzden istek yapması ve cevap alması senaryosu ile farklı bir sistemin (Örnek: Entegrasyon işlemi yapan BPMS) web servisi aracılığı ile servisten istekte bulunması ve cevap alması senaryoları bu kısımda incelenecektir.

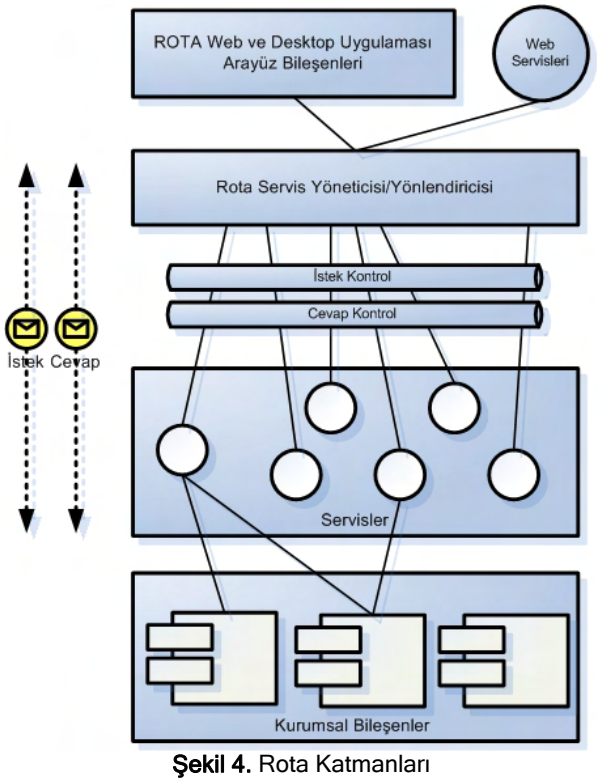
Kullanıcı Arayüzden İstek Yaparsa:

1- Kullanıcı bilgileri girer, onaylar
2- Servis yöneticisi istek mesajını yaratır
3- İstek mesajına ekrandan girilen bilgiler eklenir
4- Servis yöneticisi mesajı öncelikle istek kontrol "pipeline"a iletir.
5- 4. adım başarıyla tamamlandıktan sonra istek mesajı ilgili servise iletilir.
6- Servis mesajı işler, sonuçları üretir
7- Sonuçlar servis yöneticisine döndürülür
8- Sonuç mesajı cevap kontrol "pipeline" a iletilir
9- Cevap mesajı ekrandaki ilgili kontrole aktarılır (Servis yönetici ile bağlantıyı sağlayan yardımcı ekran bileşenleri Rota uygulama çatısı içerisinde yer almaktadır)

Dış Sistem Web Servisi Aracılığı ile İstek Yaparsa:

1- Dış sistem ilgili web servise, servisi istediği biçimde istekte bulunur
2- Web servisi servis yöneticisinden istek mesajı yaratmasını ister
3- Servis yöneticisi istek mesajı yaratır ve web servisi bu mesajı gelen bilgiler ile doldurur
4- Servis yöneticisi mesajı öncelikle istek kontrol "pipeline"a iletir.
5- 4. adım başarıyla tamamlandıktan sonra istek mesajı ilgili servise iletilir.
6- Servis mesajı işler, sonuçları üretir
7- Sonuçlar servis yöneticisine döndürülür
8- Sonuç mesajı cevap kontrol "pipeline" a iletilir
9- Cevap mesajı web servisi aracılığı ile dış sisteme aktarılır

Her iki senaryo incelendiği zaman görüleceği gibi, bir arayüzden veya bir dış sistemden istek yapılsa bile servisler ortak olarak kullanılabilmektedir. Daha önceden de belirtildiği gibi servisler birbirlerinden bağımsız ve durumsuz olarak çalışabilmektedir.



Şekil 4. Rota Katmanları

YNA Sistemi ile Varolan Sistemlerin Entegrasyonu

Daha önceden de belirtildiği gibi YNA sisteminin varolan SAP ve AS400 sistemleri ile entegre çalışması temel gereksinimlerden birisidir.

Kurumun büyüklüğü, iş hacmi ve süreçlerinin karmaşıklığına göre birbirleri ile etkileşim içerisinde olacak olan uygulamaların sayısının arttığı düşünülecek olursa bu tip entegrasyon katmanlarının farklı katmanlar olarak geliştirilmesinin önemi farkedilmektedir.

YNA, SAP ve AS400 sistemleri arasındaki entegrasyon katmanı mesajlaşma yolu mimarisine uygun hazır bir yazılım ile oluşturulmuştur.

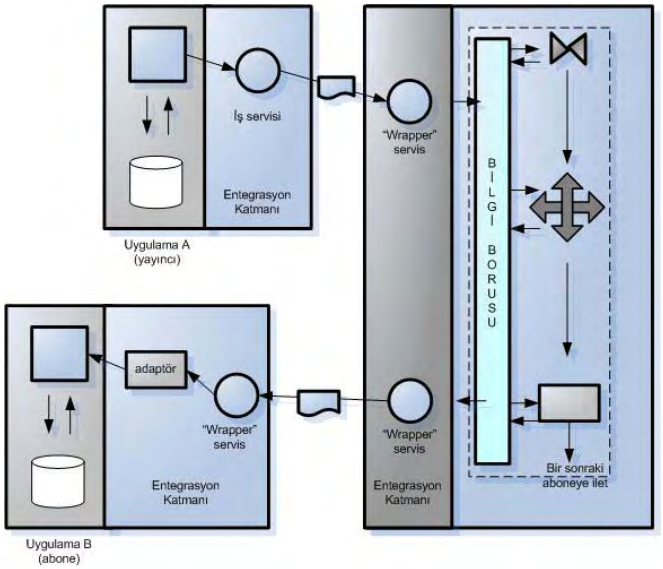
Mesajlaşma Yolu Mimarisi

Yayımcı ve Abone modeli olarak ta bilinmektedir. Genel mimari "hub-spoke" mimarisine benzemektedir, istemci uygulamaların bağlandığı merkezi entegrasyon ortamından oluşmaktadır. Veri işleme yapısı farklılık göstermektedir .

Şekil 5' te görüldüğü gibi, bu mimari gelen giden mesajları taşıyan bilgi borusunu ortaya çıkartmıştır. Uygulamalar yayımcı veya abone rollerindedirler .

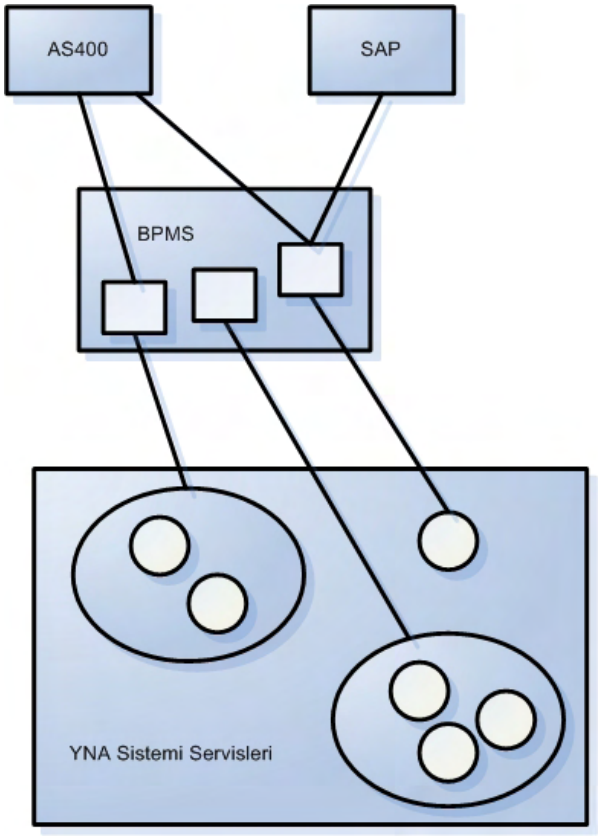
Her olası entegrasyon kaynağı diğer uygulamaların abone olabileceği bir yayımcı olarak düşünülmektedir .

Yayımcı bilgiyi gönderdiği zaman yol, veriyi her aboneye iletmektedir. Mesajlar hedefe ulaşmadan bir takım işlemden geçebilmektedir. Veri, süreç akışı sırasında, dönüştürülebilir, yönlendirilebilir, şartlı mantıklara tabi tutulabilir.



Şekil 5. Mesajlaşma Yolu Mimarisi

Tanımlanmış olan entegrasyon iş süreçleri, SAP ve AS400 sistemleri ile adaptörler aracılığı ile YNA sistemi ile web servisleri aracılığı ile konuşmaktadır.



Şekil 6. Entegrasyon iş süreçleri

9. Sonuçlar

Rota uygulama çatısı ile geliştirilen uygulamalarda, bağımsız ve yeniden kullanılabilir servisler yazılması sayesinde entegrasyon maliyetlerinde önemli bir kazanç sağlanmaktadır.

Servislerin birbirlerine bağımlı olmaması sayesinde gereksinimlerdeki değişimler daha çabuk ve güvenilir bir şekilde hayata geçirilmektedir.

Rota uygulama çatısı kullanılarak geliştirilecek olan uygulamalar, teknolojiye bağlı olmadan, web servisleri aracılığı ile birbirleri ile konuşabilecek, varolan sistemlerin erişilmek istenen operasyonları için geliştirilecek olan web servisleri ile de rahatlıkla iletişim kuracaklardır.

İleriki aşamalarda, klasik kurumsal uygulama entegrasyonu mimarileri yerine kurumsal servis yolu mimarileri kullanımı için önemli bir adım atılmıştır.

10. Kaynaklar

[1] Fehmi Hüdayioğlu, 2006, "SOA ve ESB Nedir?", http://turk.internet.com/haber/yazigoster.php3?yaziid=14659

[2] ServiceOrientation, 2006, http://www.serviceorientation.org/

[3] Selda Güner, 2005, "Architectural Approaches, Concepts and Methodologies of Service Oriented Architecture", Master Thesis, www.sts.tu-harburg.de/pw-and-m-theses/2005/gune05.pdf

[4] Enterprise SOA , 2004, The COAD Series, "Service –Oriented Architecture Best Practices", ISBN: 0-13-146575-9

[5] Service Oriented Architecture, 2004, Prentice Hall, "A Field Guide to Integrating XML and Web Services", ISBN: 0-13-142898-5

[6] Pehepe, 2006 http://www.pehepe.org/dersler.php?sayfa=2&ana=2&alt=5&dersno=7

[7] WS-Coordination, 2005, Web Services Coordination Specification, Version 1.0 August 2005 Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International Business Machines Corporation, IONA Technologies, Microsoft Corporation, Inc. http://specs.xmlsoap.org/ws/2004/10/wscoor/wscoor.pdf

[8] WS-AtomicTransaction, 2005, Web Services AtomicTransaction Specification, Version 1.0 August 2005 Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International Business Machines Corporation, IONA Technologies, Microsoft Corporation, Inc. http://specs.xmlsoap.org/ws/2004/10/wsat/wsat.pdf

[9] Thomas Earl, 2006, "WS-Standards", http://www.ws-standards.com/

[10] BPEL4WS 1.0, 2003, "Business Process Execution Language for Web Services 1.0", BEA, IBM, Microsoft http://www106.ibm.com/developerworks/webservices/library/ws-bpel/

[11] Mark Little-Thomas Freund, 2003, "A comparison of Web services transaction protocols", http://www.ibm.com/developerworks/webservices/library/ws-comproto/

[12] Thomas Freund, 2002, "An overview of WS-Transaction and WS-Coordination", http://www.ibm.com/developerworks/webservices/library/ws-wstx2

[13] Rania Khalaf, Nirmal Mukhi, Sanjiva Weerawarana, "Service-Oriented Composition in BPEL4WS", IBM T.J. Watson Research Center http://www2003.org/cdrom/papers/alternate/P768/choreo_html/p768-khalaf.htm

[14] Net.ObjectDays, 2002, "Web Services between Demand and Reality" http://www.inubit.de, 2002

Anadil İş Uygulamaları Geliştirme ve Çalıştırma Platformu

Atila Zeybek
Model Bilgi Işlem Hizmetleri Sanayi ve Ticaret Ltd. Şti.,
Yıldızposta Caddesi Ayyıldız Sitesi No 28-58,
Gayrettepe 34394 İstanbul, Türkiye
ARGE Bölümü
atila.zeybek@mbi.com.tr

Özet

Bu yayın, iş uygulaması geliştirme zorluklarını ortaya koyduktan sonra, bu zorluklara çözüm olarak, doküman merkezcil ve açık kaynak kodlu iş uygulamaları geliştirme altyapısının sahip olması gereken yapısal prensipleri ele almakta ve bu prensipler doğrultusunda geliştirilmiş olan Anadil Çerçevesi iş uygulamaları geliştirme ve çalıştırma platformu tecrübesini ortaya koymaktadır. Altyapıda iki farklı geliştirme ekibi öngörülmektedir. Teknoloji ekibi, yapısal programlama ile arayüz, veritabanı diyalogları, doküman soyutlama, hata ve güvenlik yönetimi, gibi teknik kütüphaneleri ve geliştirme aracını ortaya koymaktadır. Fonksiyon ekibi ise, teknoloji ekibinin ortaya koyduğu araç ile sadece iş akışlarına odaklanmaktadır. Doküman bazlı yaklaşım, yalın iş akışı bileşenleri ile bir programlama modeli ve disiplini öngörülmüştür.

1. Giriş

Büyük teknolojik platform değişiklikleri yazılım firmaları ve ekipleri için, her zaman zorluklar yaratmıştır. Yazılım endüstrisi, istemci-sunucu mimarilerden, web merkezli dağıtık mimariye sahip uygulama geliştirme mimarilerine doğru önemli bir teknolojik platform değişikliği sürecindedir.

Arayüz Yönetimi, Hata Yönetimi, Transaction Yönetimi, Veritabanı Tutarlılığı, Release Yönetimi gibi katmanlara gerekli önemin verilmediği ve iş akışlarının bu katmanlardan ayrı olarak ele alınmadığı yazılım projelerinde zorluklar yaşanmaktadır.

Web merkezli dağıtık platform çok daha zorlu ve büyük bir sıçramadır. Örneğin, iş uygulamalarının, servis tabanlı, event bazlı, dağıtık, mobil, mesaj tabanlı, XML bazlı, ölçeklenebilir olabilmesinin yanında, uygulama entegrasyonları da büyük önem kazanmıştır.

Bu nedenle web merkezli, dağıtık, düşük maliyetli, birlikte çalışmaya açık, açık kaynak kodlu kurumsal iş uygulamaları geliştirme platformları önem taşımaktadır.

2. Anadil Çerçevesi

2.1. Çözümü Amaçlanan Teknik Problemler

Kapsamlı iş uygulamaları geliştirmek zorluklarla doludur. Büyük yazılım ekipleri tarafından geliştirilen iş uygulamaları, yazılım mimarisi üzerinde yeterince çalışılmamış, teknik altyapı ve iş akışı odakları karışmış, teknoloji değişimlerine karşı korunmasız, belli bir disiplini öngörmeyen altyapılar üzerinde geliştirilen ve kullanıcılardan gelen değişikliklerle hantallaşan yazılımlar haline gelebilmektedir.

2.1.1. Yazılım projeleri planlanan sürelerde bitmiyor. Bilinen bir çok geliştirme altyapısı iş akışı odaklı olmadığından, iş uygulamalarında, geliştirme ekiplerini, altyapısal çalışmalar yapmak zorunda bırakmaktadır [1]. Bu nedenle iş uygulaması geliştirmek isteyen ekipler, iş akışlarını programlamaya başlayıncaya kadar, altyapı çalışmalarında uzun zaman harcamaktadırlar. Sonuçta bir çok yazılım geliştirme projesi, hedeflenen zaman planının dışına taşmakta, öngörülen maliyetlerin üzerine çıkılmaktadır.

2.1.2. İdeal bir geliştirme aracı bulunmuyor. Teknik yönü güçlü geliştirme araçları, yetkinliklerine karşın, iş akışlarını kodlamaya ve yönetmeye yatkın değildirler [2]. Gerekli güçlü bileşenlere sahip olmayan bu araçlarda, programcıların verimli hale gelmesi, uzun zaman almakta, maliyetleri yüksek kalmaktadır. Genel

amaçlı, kullanımı kolay, hızlı fakat zayıf uygulama geliştirme araçları ise, yüksek hacimlerle çalışan, büyük ölçekli firmaların, sağlamlık, güvenilirlik, performans ve kalite ihtiyaçlarını karşılayamamaktadır.

2.1.3. Geliştirme altyapısı ile iş akışları aynı katman ile yazılıyor. Kullanılan altyapılar genel amaçlı olup iş uygulamaları için tasarlanmadığından, altyapıda çözülmesi gereken bir çok teknik fonksiyon, iş akışı programlarında tekrar tekrar kodlanabilmektedir.

2.1.4. Geliştirme altyapısı ve iş akışlarını aynı ekip yazıyor. Teknik kütüphaneler ile iş akışlarının programlanması, programcı yetenekleri ve deneyimleri açısından oldukça farklı nitelikler taşımasına rağmen, aynı ekip her iki katmanı yazmaya çalışıyor. Sonuçta teknolojik değişim iş akışlarını, iş akışlarındaki değişim altyapıyı olumsuz etkilemektedir.

2.1.5. Altyapılar değişiklik yönetiminde esnek değiller. Kurumsal Kaynak Planlama gibi uygulamalarda, orta ve büyük ölçekli firmaların, iş akışlarında değişiklik talepleri kaçınılmazdır. Bu değişikliklerin hızlı ve güvenli bir şekilde yapılamaması, altyapılarda, fonksiyonel değişiklik yönetiminin tasarım aşamasında ele alınmamış olmasından kaynaklanmaktadır.

2.1.6. Teknoloji değişim dalgalarında iş akışları yeniden yazılıyor. Uygulama geliştirme ve çalıştırma ortamlarında, ortalama beş yılda bir gerçekleşen teknolojik gelişmeler, uygulamaların iş akışlarının ve teknik kütüphanelerinin yeniden yazılmasını gerektirmektedir. Bu durum yazılım firmalarını ve ekiplerini verimsiz kılmakta, kullanıcılara büyük problemler yaşatmaktadır.

2.1.7. Bir programlama modeli ve disiplini bulunmamaktadır. Geliştirme ortamları, yazılım geliştirme ekiplerine ortak bir programlama modeli ve disiplini sağlamadığından, eleman sirkülasyonlarında, programcılar için, bir başkasının yazdığı programları anlamak, adapte olmak, kavramak ve istenen değişikleri yapmak, hataları düzeltmek zorlaşmaktadır.

2.1.8. Kullanımı ve öğrenimi zor arayüzler. Her programcı ekip, yeni arayüzler geliştirdiğinden, zaman kısıtları ve belge bazlı arayüz sistemlerinin tasarım zorlukları nedenleriyle, kullanımı ve öğrenimi zor, iş uygulama arayüzleri ortaya çıkmaktadır.

2.1.9. Birlikte çalışabilme zorlukları. İş akışı odaklı, belge merkezcil ve bileşen bazlı bir altyapının eksikliği, iş uygulaması projelerinde, bileşenlerin tasarlanmasında, paylaşılmasında, programlama, test, destek ve versiyon yönetim aşamalarında zorluklara neden olmaktadır. Altyapılar bileşen bazlı belli bir programlama modeli ve disiplini öngörmediğinden, farklı coğrafyalarda bulunan iş ortakları ile ortak projelerde çalışmak zorlaşmaktadır.

2.1.10. İş akışları açık kaynak kodlu değil. Müşteride, iş akışlarında ihtiyaç duyulan değişiklikleri, geliştirici firma - çözüm ortağı – müşteri yazılımcı ekibi zincirinde, istenilen seviyede ele alabilme özgürlüğü bulunmamaktadır.

2.1.11. Açık kaynak kod ve gelir sağlayıcı iş modeli mümkün olamamaktadır. Mevcut altyapılar, yazılım evleri, geliştirici iş ortakları ve müşteri yazılım ekipleri zincirinde, açık kaynak kodlu iş uygulamaları geliştirme, iş uygulamalarını koruma, lisans satışı, dağıtma, paylaşma, destekleme faaliyetlerini kapsayan gelir sağlayıcı bir iş modeli oluşmasına yardımcı olmamaktadır.

2.2. Hedeflenen Prensipler

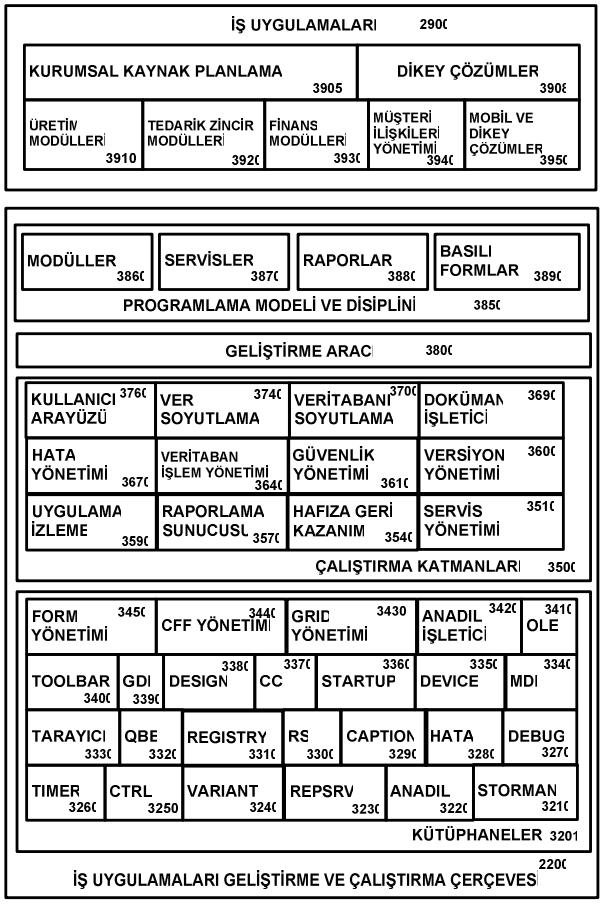
İş uygulaması geliştirme zorluklarını ortadan kaldırmak üzere tasarlanan Anadil Çerçevesi Yazılım Mimarisinde, aşağıdaki tasarım prensipleri hedeflenmiştir.

1. Altyapı iş uygulaması odaklı bir altyapı olmalıdır.
2. Altyapı, teknoloji katmanı ile iş akışları üzerinde iki farklı ekibi desteklemeli. Uygulama ekibi sadece iş akışları üzerinde çalışarak uygulama geliştirebilmelidir.
3. Teknoloji katmanı iş akışlarını desteklemeli fakat iş akışlarından bağımsız olmalıdır. Teknoloji değişim dalgalarında iş akışları korunabilmeli yeniden yazılmamalıdır.
4. Altyapı bir programlama modeli ve disiplini öngörmelidir.
5. Güvenilir ve sağlam uygulamalar küçük ekiplerle kolay ve hızlı geliştirilebilmelidir.
6. İş akışları bileşen bazlı ve açık kaynak kodlu olmalıdır.
7. Bileşen tipleri yalın ve az sayıda olmalı, iş uygulaması bileşen tiplerini ifade etmelidir.
8. İş akışları üzerinde çalışan programcılar diğer programcıların kodlarına kolay uyum sağlayabilmelidir.
9. Belgeye ait bilgiler form ile veritabanı arasında soyutlanabilmeli ve bu yapı, fonksiyon ekibinin, iş akışlarına şekil verebilmesini kolaylaştırmalıdır.

10. Kullanıcı programa değil belgeye erişmelidir. Kullanıcı aynı anda birden fazla belge ile çalışabilmelidir. Altyapı belge bazlı arayüz etkileşimini yönetmelidir.
11. Uygulama bileşenlerine, altyapının sunduğu, uygulama bağımsız tarayıcı üzerinden erişilmelidir.
12. İş akışlarında yapılan bir değişiklik, altyapıda yeniden derlemeye neden olmamalıdır.
13. Altyapı, iş akışlarında gerçekleşen veritabanı diyaloglarını ve bağlantı havuzunu yönetmelidir. Altyapı, uygulama ve kullanıcı güvenlik yönetimini yapmalıdır.
14. Altyapı, katmanlarda meydana gelen hataları yönetmeli ve kullanıcıya düzeltme şansı tanımalıdır.
15. Altyapı, iş akışlarının debug edilmesine ve katmanlarda log yapılmasına izin vermelidir.
16. Uygulama katmanında, bir iş akışı bileşeni diğer iş akışı bileşeninin, servislerini çağırabilmelidir. Uygulama katmanında, bir uygulama bileşeni diğer bileşeni gömülü olarak kullanabilmelidir.
17. Güncellemeler bileşen bazında olmalı ve çalışma anında bir bileşen güncellenebilmelidir.

Bu prensipleri baz alan, Anadil Çerçevesi, iş akışlarından bağımsız, açık kaynak kodlu, bileşen tabanlı iş uygulamaları geliştirme ve çalıştırma platformudur. Anadil Çerçevesi, güvenlik yönetim katmanı, yardımcı kütüphaneler ve çalıştırma katmanı üzerinde, doküman bazlı kullanıcı arayüz yönetimini içermekte ve Türkçe komutlar ile çalışan, iş uygulaması odaklı Anadil Programlama Dili'ne sahip geliştirme ortamı sağlamaktadır. Anadil iş odaklı programlama disiplinini zorunlu kılmaktadır. Kullanıcılar iş yazılımı uygulama arayüzü ile etkileşirken, altyapı iş akışlarından bağımsız olarak, kullanıcı etkileşimini koordine eder, veritabanı işlemlerini ve bağlantıları yönetir, hataları ele alır, iş ve altyapı katmanlarından loglama yapar.

Geliştirme aracı Imagine ile Modül, Rapor, Servis ve Basılı Formlar geliştirilmektedir. Imagine ile geliştirilen uygulamalar, altyapı üzerinde yorumlayıcı tarafından çalıştırılırlar. Altyapı gelişmiş bir kullanıcı arabirimi, hata yönetimi, güvenlik, veritabanı soyutlama katmanı ve Cff (Context Free Format) katmanları ile bir çok iş uygulamasında ortak bulunan teknolojik işlevleri, iş akışlarından bağımsız bir şekilde yönetmektedir. Bununla birlikte teknolojik altyapı, fonksiyonel bölümler ile içiçedir. Modüller ile ilgili görsel veya fonksiyonel olayları teknolojik katman yönetirken, bu yönetim fonksiyonel akıştan tam olarak bağımsızdır. Fonksiyonel akışı programlayan programcılar, hazır olarak sunulan katmanların üzerinde, sadece kendilerinden istenilen iş akışına



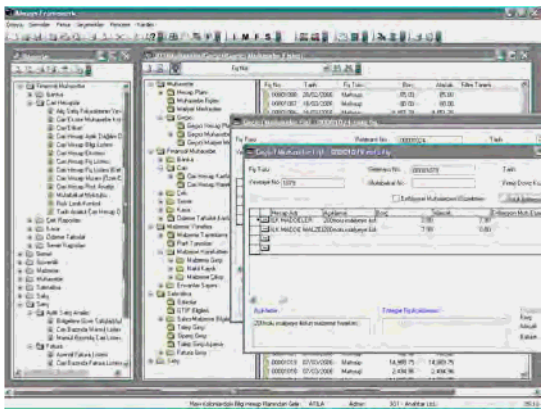
Şekil. 1. Anadil çerçevesi mimarisi.

odaklanmakta, bu sayede, kapsamlı, kaliteli ve güvenilir projeler, daha kısa zamanda üretilebilmektedirler.

2.3. Doküman Bazlı Kullanıcı Etkileşimi

Anadil Çerçevesi iş uygulaması odağı nedeniyle doküman merkezcil mimariye sahiptir. Kullanıcı, yaygın yaklaşımlarda olduğu gibi, ilgili programı açıp daha sonra ilgili kayıtlara ulaşmak yerine, ilgili dokümana ulaşılır ve altyapı bu dokümanı canlandıracak modülü başlatır. Bu yaklaşım ile gerçek belgeler ile yapılan fiziksel çalışma şekli örnek alınarak, öğrenim ve kullanım kolaylığı hedeflenmiştir.

Belgeler iç yapıda Cff ile temsil edilirler. İş akışlarının kodlandığı katman ile altyapının tüm yönetim katmanları belge yönetimi ve belge için programlama odaklı tasarlanmışlardır. Burada da amaç, kullanıcıya sağlanan öğrenim ve kullanım kolaylığının, hem teknik ekibe hem de fonksiyon ekibine,



Şekil. 2. Doküman bazlı kullanıcı arayüzü.

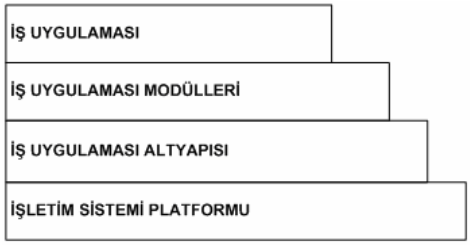
programlama, kavrama, uyum sağlama ve değişiklik yapma kolaylığı sağlamak olmuştur.

Altyapı, iş uygulamasından bağımsız, bir ana pencere içinde, modül ve raporların ağaç yapısında gösterimi için, MDI (Multiple Document Interface – Çoklu Doküman Arabirimi) yapısında iki temel pencere sunar. Modül penceresi iki bölümden oluşur: Klasör bölümü belge klasörlerini ağaç yapısında hiyerarşik olarak listelerken belge bölümü, ilgili klasöre ait belgeleri, başlık bilgileriyle listeler. Bu liste üzerinde istenilen kolon üzerinde sıralama ve arama yapılır. Çift tıklama ile ilgili belge, kendisini oluşturan modül ile açılır, değişiklik sonrası ana toolbardaki sakla tuşu ile saklanır.

Bir modül bir belgeyi temsil eder. Aynı modül, birden fazla belgeyi aynı anda destekleyebilir. Altyapı ihtiyaç oluştuğunda ilgili belgeyi canlandırır ve modüle ait doküman sayısı sıfıra düştüğünde, ilgili modülü devreden çıkarır.

Altyapı, arabirim fonksiyonlarını tüm belgelerin kullanımına sunar. Bu sayede, yeni bir modül-belge geliştiren programcının, artık, kullanıcıya belgenin gösterimi, erişimi, ürün, müşteri ve benzeri seçim işlemleri, saklama yöntemi gibi süreçlerde, çaba sarfetmesine gerek kalmamakta ve sadece iş akışının programlanmasına yoğunlaşması sağlanmaktadır. Kullanıcı arabirim etkileşimi her belge için aynı standartlar ile yönetildiğinden, bir belgeyi açıp bilgi girişi veya değişikliği yaptıktan sonra saklayan veya ilgili belgeyi silen kullanıcı, tüm belgelerin kullanımını öğrenmiş olur. Altyapı, herhangi bir iş akışı ile ilgili program koduna sahip değildir.

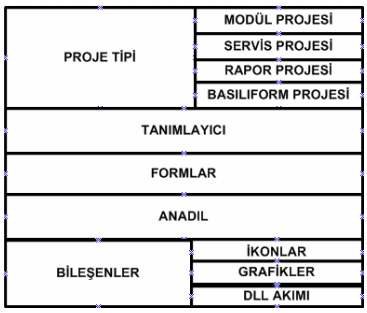
İş akışlarının programlanmasından, Anadil projeleri sorumludur ve sonraki sürümlere uyumlu kalabilmek için, Anadil projeleri, işletim sistemi programlama arabirimlerine doğrudan erişmemekte, altyapı arabirimlerini kullanmaktadır.



Şekil. 3. Anadil iş uygulaması katmanları.

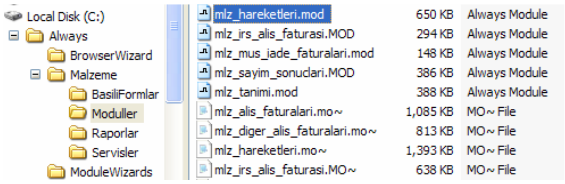
2.4. Bileşen Bazlı İş Uygulamaları Geliştirme Altyapısı

Teknoloji ve iş katmanlarını, birbirlerini destekler şekilde ayırmanın yanında, iş uygulamalarını daha küçük birimler halinde tasarlayabilmeli, bu küçük birimleri fonksiyon ekiplerine paylaştırabilmeliyiz. Bu nedenle Anadil Çerçevesi Modül, Servis, Rapor ve Basılı Form olarak dört tip Anadil Projesi sunar. Tüm Anadil projeleri altyapının sunduğu geliştirme aracı olan Imagine ile geliştirilirler.



Şekil. 4. Anadil proje tipleri

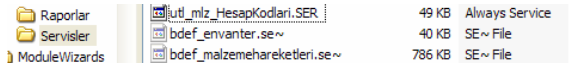
2.4.1. Modüller. Fonksiyon ekibi her bir belgeye karşılık bir modül tasarlar. Satış siparişi belgesi, muhasebe fişi belgesi, malzeme kartı belgeleri modüllere örnek olarak sayılabilirler. Bir modülün bir veya birden fazla formu olabilir, fakat bir Anadil programına sahiptir.



Şekil. 5. Malzeme yönetimi modül projeleri

*.MOD uzantılı modül projeleri ayrı dosyalar halinde AnadilRoot altında yer alır. MO~ uzantılı dosyalar altyapı tarafından yorumlanmış modüllerdir. Bu sayede kullanım anında, modül, servis veya raporların yeni sürümleri devreye alınabilmektedir.

2.4.2. Servisler. İş akışlarında tüm Anadil projeleri tarafından ortak kullanılan fonksiyonlar servislerde yer alır. Kullanıcı ile etkileşen servisler, formlara sahip olabilirler. Bazı servisler uygulama açılırken bir kez çalışırlar ve dururlar. Bazıları çalışır, aktif kalır ve çağrı bekler. Diğerleri bir başka modül veya servis tarafından gerekli olduğunda aktifleştirilebilirler. Bir Anadil servisi, altyapının uygulama sunucusu sayesinde, uzak uygulamalara web servisi olarak hizmet verebilir. Kur farkı servisi, maliyet servisi, çoklu borç kapatma servisi, belge kaydetme ve okuma servisleri, iş akışlarında kullanılan servislere örnek olarak verilebilirler.



Şekil. 6. Anadil servis birimleri

*.SER uzantılı servis projeleri ayrı dosyalar halinde AnadilRoot altında yer alır.

Servis Fonksiyonları. Servisler ServiceMain, StartService ve StartWizard olmak üzere, üç temel fonksiyona sahiptir.

```
Fonksiyon ServiceMain(Service As Object) As Bool
    srv:=Service
    srv.ExposeServiceCall("DokumanKaydet", "GlbObj")
    srv.ExposeServiceCall("YevmiyeNumarala", "GlbObj")
    srv.AddDependency( "ÖnceBaşlayacakFonksiyon")
    ServiceMain:=True
FonksiyonSonu
```

StartMain fonksiyonu altyapı tarafından tarama sırasında çalıştırılır. StartMain fonksiyonunda diğer Anadil projelerinin kullanımına sunulan fonksiyonlar ExposeServiceCall metodu ile altyapıya bildirilir. Bu fonksiyon içinde, varsa servisin bağımlı olduğu servis AddDependency metodu ile belirtilir.

```
Fonksiyon StartService(Service Tipi Nesne, Cci
Tipi Nesne) As Bool
  SetNull(@nullst)
  StartService:=Doğru
FonksiyonSonu
```

ServiceMain ile altyapı tarafından bilinir duruma gelen servis, servis kimliğinde, açılış anında başlatılsın denilmişse veya ServislerinGörünümü servisinden başlatıldığında veya bir başka servis veya modülden bu servise çağrı yapıldığında, StartService çalıştırılır. StartService bir kez çalışır. Kullanılan parametrelere StartService içinde ilk değer verilebilir, menü, tuş veya toolbar eklenebilir.

```
Fonksiyon StartWizard(Service Tipi Nesne) Tipi
Mantıksal
  DialogBox("f", 0, 0)
  StartWizard:=Doğru
FonksiyonSonu
```

Servise bir çağrı yapıldığında, StartWizard fonksiyonu çalıştırıldıktan sonra ilgili fonksiyon tetiklenir.

```
Fonksiyon f_Open(f Tipi Nesne) Tipi Mantıksal
  f.CenterForm()
  f_Open:=Doğru
FonksiyonSonu
```

2.4.3. Raporlar ve Basılı Formlar. Benzer şekilde *.REP uzantılı rapor projeleri ve *.FRM uzantılı basılı form projeleri kendi klasörlerinde yer alırlar.

2.4.4. Sihirbazlar. Altyapı yine Anadil programlama dili ile yazılmış sihirbaz servis kodları ile, basit modül, gridli modül, basit servis, tarayıcı servisi ve rapor projelerini hiç kod yazmadan oluşturabilme şansı vermektedir. Fonksiyon ekibi yeni sihirbazlar ortaya koyabilmektedir.

Modül, Servis, Rapor ve BasılıForm bileşenlerinin her birinin, disk üzerinde, AnadilRoot altında fiziksel bir dosya olarak yer alıyor olması, yazılım ekiplerine önemli esneklikler sağlamaktadır. Örneğin yeni geliştirilecek, üretim veya bakım onarım gibi ana modüllerin, küçük parçaları ekip içinde, yetkinliklere göre paylaştırılabilmektedir. Bir bileşende değişiklik yapılması, bir altyapı veya fonksiyon bileşeninin yeniden derlenmesini gerektirmemektedir. Kullanıcıya, müşteriye ve sektöre özgü güncellemelerin yapılabilmesi ve desteklenebilmesi mümkün olabilmektedir. Ayrıca, çözüm ortağı, yeni bir dikey sektör ürünü ortaya çıkarmak istediğinde, mevcut bileşenlerin üzerine, eksik gördüğü servis ve modülleri geliştirip, düşük bir maliyet ile yeni ürünü piyasaya sunabilmektedir.

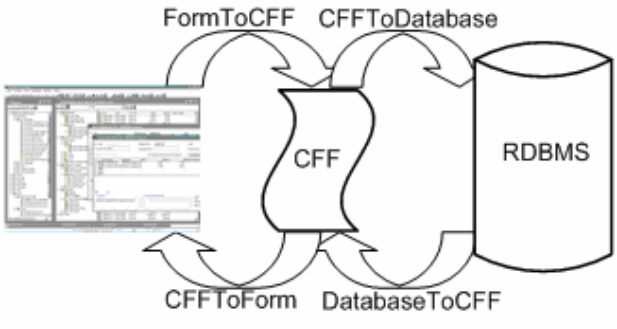
2.5. Arayüz ile Veritabanı Arasında Veri Soyutlama : CFF

Fonksiyon ekiplerinin iş akışlarını etkin bir şekilde programlayabilmesi için, belge bilgilerine, veritabanı ve form nesneleri dışında kolayca erişiyor, değişiklik yapabiliyor ve paylaştırabiliyor olabilmeleri gerekmektedir. Bu amaçla CFF (Context Free Format :

Serbest Bağlamlı Veri Yapısı) adlı bir veri taşıma yöntemi ortaya konmuştur.

CFF veri taşıma işlemlerinde kullanılır. Yapı olarak bünyesinde veri tutmakta olan CFF nesneleri ağaç şeklinde bir organizasyona sahiptirler. Anadil uygulamalarında veritabanından alınan bilgilerin kullanıcının karşısına gelmeden istemci bazında bir yerden bir yere taşınması görevi CFF nesneleri tarafından başarılmaktadır.

Ağaç şeklinde organizasyonunda CFF nesneleri başka alt CFF nesnelerine sahip olabilmektedir. Dinamik yapıları nedeniyle veri tutma kapasitelerini genişletebilmektedirler. CFF nesnelerinin Anadil altındaki temel işlevi, veritabanı sunucusu ile kullanıcı arayüzü arasındaki bilgi akışında bir soyutlama katmanı sağlamaktır.



Şekil. 7. Doküman Saklama ve Erişimi

2.6. Programlama Modeli ve Disiplini.

Anadil Çerçevesi'nde belgelerin, kullanıcı arayüzü ile veritabanı arasındaki işlemlerinde dört temel iş akışı fonksiyonu görev alır. Bunlar sırasıyla FormToCFF, CFFToDatabase, DatabaseToCFF ve CFFToForm dur. Bu fonksiyonların içini belgelerin akış ihtiyaçlarına göre, fonksiyon programcısı doldururken, tüm mekanizmayı altyapı yönetmektedir. Bir başka deyişle, teknoloji ve fonksiyon hem içiçedir, hem de kesin çizgilerle ayrılmıştır.

2.6.1. FormToCFF. Kullanıcı herhangi bir belgeyi (Malzeme Giriş Dokümanı, Muhasebe Fişi, Sipariş, Fatura,...) doldurup ana toolbardaki Sakla tuşuna bastıktan sonra sırasıyla tetiklenen olaylar şunlardır:

Öncelikle ilgili modülün FormToCFF fonksiyonu tetiklenir. Örneğin modülümüz MusteriSiparisi.MOD olsun. Bu durumda MusteriSiparisi.MOD içerisinde bulunan MusteriSiparisi_FormToCFF isimli fonksiyon çalıştırılır:

```
Fonksiyon   MusteriSiparisi_FormToCFF(f   Tipi
   Nesne,  Cff Tipi Object) As Bool

   'Form bilgileri cff e aktarılıyor
   Cff.add("SiparisNo", f.SiparisNo.ValueStr)
   Cff.add("MusteriNo", f.MusteriNo.ValueStr)
   Cff.add("SipTarihi", f.SipTarihi.ValueDate)

   '...Diğer başlık alanları...

   'Sipariş satırları için alt CFF yaratılıyor
   SubCFF:=Cff.Create("Rows")

   'Tüm sipariş satırları tek bir method ile
   'cff e alınıyor
   f.SiparisGrid.GetRows(SubCFF)

   MusteriSiparisi_FormToCFF:=Doğru
FonksiyonSonu
```

FormToCFF adımı ana proses bünyesinde çalışır. Bu aşamada formda bulunan tüm bilgiler CFF yapısına alınmış olur. FormToCFF akışı, organizasyonun işleyişine göre, her belgede farklılık gösterebilir. İş akışını bu bölüme yansıtmak fonksiyon programcısının görevidir. Kullanıcı arayüzü ile veritabanı arasında bilgilerin CFF ortamına alınması, programcılara, iş akışlarına istenildiği gibi hükmetme kolaylığı vermekte, iş akışı programlamaya ortak bir disiplin ve kalite getirmektedir. Formlardan satır bilgilerinin alınması her programcıyı her bir modülde ayrı ayrı uğraştıran bir zorluk olmuştur. Grid.GetRows metodu ile Anadil programcılara yardımcı olmaktadır. Kullanıcı grid üzerinde işlemler yaparken, altyapı yeni eklenen satırları Inserted, silinenleri Deleted ve güncellenenleri Updated alt cff lerinde takip eder. GetRows metodu da verilen CFF in altına bu üç alt cff i ilgili satır ve kolon bilgileri ile birlikte ekler. Bu sayede iş akışı programcısı tek method ile gridlerden eklenen, silinen ve güncellenen satırları gruplar halinde eline alır ve istenilen akışı programlar.

2.6.2. CFFToDatabase. İkinci adımda altyapı, CFFToDatabase fonksiyonunu tetikler:

```
Fonksiyon MusteriSiparisi_CFFToDatabase(Cff as Object,
IsNewDoc As Bool, r_sayac Tipi Sayı) Tipi Mantıksal
  Değişken rs Tipi Nesne  'RS nesnesi

  Eğer IsNewDoc ise
        Subcff:=cff.OpenSubCFF("Rows")
        Insertedcff:=Subcff.OpenSubCFF("Inserted")
        Updatedcff:=Subcff.OpenSubCFF("Updated")
        Deletedcff:=Subcff.OpenSubCFF("Deleted")
        'Başlık Alanları
        rs:=BuildInsertStm("sts_mus_siparisi")
        rs.AddInsertField("Toplam", cff.toplam)
        rs.AddInsertField("Siparis_Tipi", cff.siparis_tipi)
        rs.CompileStm()
        rs.ExecuteStm()
```

```
'Yeni doküman numarası altyapıya veriliyor
r_sayac:=rs.r_sayac

'Yeni Eklenen Satırlar
Insertedcff.MoveToFirstRow()
Eğer InsertedCff.MoveToFirstEntry() ise
  Döngü
     rs:=BuildInsertStm(  "sts_mus_siparis_detay")

     'Doküman başlık-satır ilişkisi
     rs.AddInsertField( "mus_rsayac",r_sayac)
     '...
     rs.compilestm()
     rs.executestm()
     rs.close()
  DöngüSonu Oldukça
        Insertedcff.MoveToNextRow()
 EğerSonu

 'Güncellenen Satırlar
 Updatedcff.MoveToFirstRow()
 If UpdatedCff.MoveToFirstEntry() then
   do
     rs:=BuildUpdateStm( "sts_mus_siparis_detay")
     '...
     rs.CompileStm()
     rs.ExecuteStm()
   loop while Updatedcff.MoveToNextRow()
 endif

 'Silinen Satırlar
 if  DeletedCFF.MoveToFirstRow() Then
     do
        rs:=BuildDeleteStm(
             "sts_mus_siparis_detay")
        rs.AddWhereConJunct("r_sayac",
             DeletedCff.row_id)
        rs.CompileStm()
        If Not rs.ExecuteStm() Then
           wrk_muhfis_CFFToDatabase :=Yanlış
           Return
        EndIf
        rs.Close()
   loop until not DeletedCFF.MoveToNextRow()
 endif
 EğerSonu
 MusteriSiparisi_CFFToDatabase:=Doğru
FonksiyonSonu
```

CFFToDatabase fonksiyonu transaction kapsamındaki alt proses bünyesinde çalıştırılır. Bu nedenle kullanıcı tarayıcıda çalışmalarına devam edebilir. CFFToDatabase fonksiyonu yine altyapı tarafından kolaylıkla gerçekleştirilebilir. Fakat burada da hedef, iş akışlarını fonksiyon programcısının yazmasıdır.

2.6.3. DatabaseToCFF. Belgelerin veritabanından formlara akış yolunda, bu kez yukarıdaki işlemlerin tam tersi uygulanmaktadır :

```
cff.add("SiparisNo",  rs. SiparisNo)   'DbToCff
f.SiparisNo.ValueStr := cff.SiparisNo 'CFFToForm
```

Bu işleyiş, kullanıcılar ile veritabanı arasında, yaratılan, düzeltilen ve silinen tüm belgeler için aynı şekilde ele alınmaktadır. Bu yaklaşım ile Anadil, fonksiyon ekiplerine ortak bir programlama modeli, disiplini ve model bazlı mimari öngörmektedir.

2.7. Katmanlarda Dokümanların Ele Alınışı

2.7.1. Uygulama Tarayıcı Klasörlerinin Oluşturulması. Uygulama, başlatıldığında, AnadilRoot altında bulunan, kapsam dahilindeki işletim sistemi klasörlerini taramaya başlar. Her modül grubu, bdef_ModulIsmi.SER benzeri Anadil servis dosyalarına (bdef : Browser Definition Service : Tarayıcı Tanım Servisi) sahiptir. Bdef servisleri, tarayıcı ağaç yapısında, modüllerin klasör yapılarını tarif ederler.

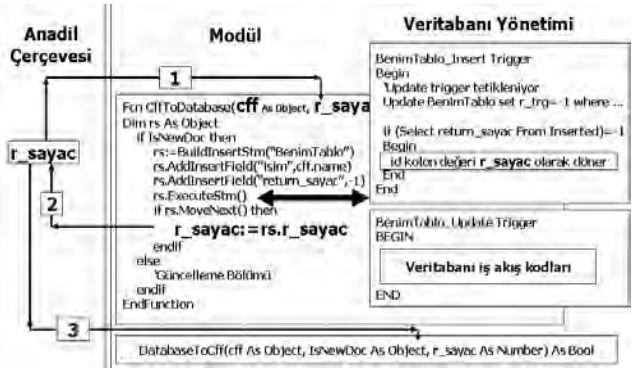
Tarayıcı servislerinde bulunan AddBrowseFolderEx çağrıları ile klasörler oluşturulur. CreateColDef komutları ile de ilgili klasörde yer alacak kolonlar tarif edilirler. Kolonlar ilgili dokümanın tarayıcıda görülen, sıralanan ve arama yapılabilen başlık bilgileridir. Kolon tanımında bulunan SQL ifadesi, ilgili kolonda sıralama yapılmak istendiğinde altyapı tarafından uygulanacak SQL tümcesini belirler. Arama alanına girilen kriter ile arama tuşuna basıldığında ise SearchSQL tümcesi çalıştırılır. İş akışlarını ortaya koyan ekip, tarayıcı tanım servisleri aracılığı ile, tarayıcı klasör yapısını, klasör-belge ilişkisini, sıralama ve arama işlemlerinde uygulanacak SQL tümcelerini, altyapıya tanıtmaktadır. Tarayıcı tanım servisleri sayesinde, fonksiyon programcısı, kullanıcı arayüzünün ne şekilde olacağına, altyapıdan bağımsız bir şekilde karar verir. Altyapı da kendisine tarif edilen arayüz yapısını, iş akışlarından bağımsız bir şekilde yönetir.

```
Fonksiyon ServiceMain(Service As Object) As Bool
  Service.AddDependency("fknt")
  Service.ExposeServiceCall("TarayıcılarıYarat", "")
  ServiceMain := TRUE
FonksiyonSonu

Fonksiyon StartService(Service As Object, Cci As Object) As Bool
  TarayıcılarıYarat()
  StartService:=Doğru
FonksiyonSonu
```

```
Fonksiyon TarayıcılarıYarat()
  Değişken cd  Tipi Nesne
  Değişken Srv  Tipi Nesne
  Değişken view Tipi Mantıksal
  ' BWIZARDSTART: Malzeme Yonetimi
  ' Tarayıcı sihirbazı tarafından yaratılan kod. Değiştirmeyin!
   cd:=CreateColDef()
    AddBrowseFolderEx( GetFolderFromStructuredName(""),#
              "Malzeme Yönetimi",#
              "","",cd,FALSE,"12.00")

  ' BWIZARDEND: Malzeme Yonetimi
  ' BWIZARDSTART: /Malzeme Yonetimi/Parti Tanimlari
  ' Tarayıcı sihirbazı tarafından yaratılan kod. Değiştirmeyin!
   cd:=CreateColDef()
   cd.Name:="Malzeme Kodu"
   cd.SQL:="Select mlz.r_sayac rs,
              mlz.malzeme_kodu rs2,p.malzeme_adi
         from mlz_ana_parti_tanimi mlz, mlz_tanimi p
                    where   p.malzeme_kodu=mlz.malzeme_kodu
                    and p.cid=mlz.cid and mlz.cid = (?) order by
                    mlz.malzeme_kodu"
   cd.SearchSQL:="Select  mlz.r_sayac  rs,  mlz.malzeme_kodu
                  rs2,p.malzeme_adi  from  mlz_ana_parti_tanimi
                  mlz,mlz_tanimi  p  where
                  p.malzeme_kodu=mlz.malzeme_kodu  and
                  mlz.malzeme_kodu>=(?) and p.cid=mlz.cid and
                  mlz.cid = (?) order by mlz.malzeme_kodu"
   cd.Field:="rs2"
   cd.Sortable:=True
   cd.Format:="s"
   cd.CounterField:="rs"
   cd.Type:=VARTYPE_TEXT
   cd.DisjunctService1:="fknt"
   cd.DisjunctServiceCall1:="GetCurrentCompany"
   cd.DisjunctServiceParam1:="CompanyId"
   cd.AddTitle()
```

Şekil. 8. Bdef_MalzemeYonetimi.SER tarayıcı kolon tanım örneği.

2.7.2. Yeni bir Belgenin Saklanması. Kullanıcıya dokümanlarla çalışma arayüzü sunan altyapılarda, tüm katmanlar aynı yaklaşıma sahip olmalıdırlar. Kullanıcının yeni doküman talebine karşılık, altyapı ilgili dokümanın modülünü başlatır. Modül kullanıcının karşışına boş bir belge formu getirir. Veri girişlerini tamamlayan kullanıcı sakla tuşuna basar. Altyapı FormToCff fonksiyonunu tetikler. İş akışı katmanı, form üzerindeki bilgileri Cff nesnesine aktarır. Henüz belge veritabanına saklanmadığından, r_sayac belirsizdir. FormToCff fonksiyonundan sonra ilk adımda Cff ve r_sayac parametreleri ile CffToDatabase fonksiyonu tetiklenir. Sunucuya dönük bağımsız bir proses olarak çalıştırılan bu iş akışı fonksiyonunda, Cff bilgileri Rs nesnesine aktarılır. Rs nesnesi içine, oluşacak doküman numarasının geriye istendiğine dair return_sayac alanı eklenir. Ve rs.execute metodu çalıştırılır. Bu aşamada, altyapının veritabanı katmanı devreye girer. Insert Trigger sonrasında, sunucu tarafındaki iş akışları Update Trigger bünyesinde uygulandıktan sonra kontrol, Insert Triggerda kaldığı noktaya geri döner.



Şekil. 9. Doküman numarasının katmanlarda ele alınışı.

Artık ilgili belgenin üst bilgileri başarıyla saklanmış ve bir belge numarası otomatik olarak oluşmuştur. Insert Trigger'ın son bölümünde yeni oluşan doküman numarası r_sayac olarak altyapıya geri döndürülür. Bu bilgi altyapı tarafından, Anadil iş akışları seviyesine rs.r_sayac olarak taşınır ve kontrol Anadil rs.executestm() satırından bir sonraki satıra ilerler.

İkinci aşamada r_sayac:=rs.r_sayac komutu ile yeni yaratılan dokümanın belge numarası altyapıya aktarılmış olur. Bu atamada rs nesnesinden bilgi alırken, Java kodlarının aksine [4], veri tipleri ile ilgili herhangi bir işaret veya method kullanılmamıştır. İş uygulaması geliştirme odağına sahip Anadil Çerçevesi, fonksiyon programcılarına iş uygulamaları için gerekli olan, Sayı, String, Mantıksal ve Nesne gibi yalın veri tiplerini sunarak, programcıları iş akışlarına yönlendirmektedir. Böylece programcı, hangi değişkenin, rs, grid veya cff parametresinin, byte, short, int, float, decimal, double gibi veri tiplerinden hangisine sahip olduğunu hatırlamaya zorlanmaz. İlgili değişkenin, rs, grid veya cff üyesinin veri tipini, Anadil Çerçevesi zaten bilmektedir. Atama işlemlerini bu bilgi dahilinde sorunsuzca gerçekleştirir.

Henüz tüm transaction tamamlanmamıştır. CffToDatabase fonksiyonunun devam eden kodlarında, aynı transaction içinde, varsa ilgili belgenin satır bilgileri de saklanacaktır. Satır bilgileri ilgili tablolara saklanırken ilişkiler yine r_sayac üzerinden kurulacaktır.

CffToDatabase fonksiyonu başarıyla sonuçlandıktan sonra, kullanıcının karşısında saklanmakta olan belgeyi tekrar veritabanından okumak ve arayüzde ilgili belgeyi tazeleyerek kullanıcıya saklama işleminin sonuçlandığı bilgisini fiziksel olarak ta göstermekte fayda vardır. Bu nedenle üçüncü adımda DatabaseToCff fonksiyonu tetiklenir. Bu aşamada altyapı ilgili aktif belgenin r_sayac bilgisini yani doküman numarasını bildiğinden,

r_sayac bilgisini bu fonksiyona parametre olarak verir. DatabaseToCff fonksiyonu da kendisine doküman numarası verilen dokümanı veritabanından talep eder ve doküman bilgilerini Rs nesnesinden Cff nesnesine aktarır. Daha sonra da CffToForm fonksiyonu tetiklenecek ve ilgili belge tazelenecektir.

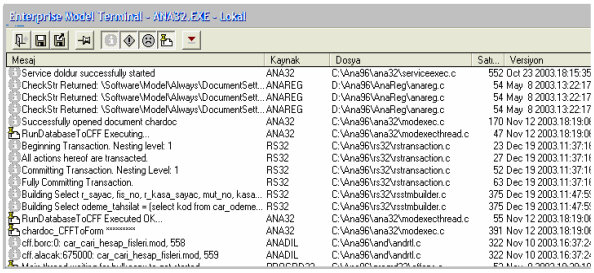
Kullanıcı ile veritabanları arasında işlemlere konu olan belgelerin, CFF ile soyutlanabilmesi ve r_sayac doküman numarası ile katmanlar arasında takip edilebilmesi sayesinde, altyapı, iş akışlarından ve belgelerden bağımsız belge bazlı yönetim yapabilmektedir.

2.7.3. Mevcut bir Belgenin Açılması. Kullanıcı tarayıcı üzerinde herhangi bir klasöre tıkladığında, altyapı, klasörün ilgili olduğu dokümanlardan, kullanıcının görebileceği sayıda dokümanı başlık bilgileri ile birlikte tarayıcıda listeler (Şekil.2.). Altyapı bu listeyi oluştururken belgelerin başlık bilgilerinin yanında, r_sayac bilgilerini de veritabanından almış olur. Bu aşamada kullanıcı dilediği başlık bilgisinde sıralama ve arama yapabilir. Kullanıcı, tarayıcı üzerinde, arayıp ulaştığı belgeye tıklayarak, altyapıya "Şu doküman numarasına sahip dokümanı aç" talebini iletmiş olur. Altyapı r_sayac bilgisi verilmiş belgeyi, belgenin ilgili modülünden talep eder. İlgili modül, DatabaseToCff fonksiyonu ile altyapının Rs katmanı üzerinden, veritabanından r_sayac numaralı dokümanı talep eder. Rs üzerinden modül ortamına alınan doküman bilgileri Cff'e aktarılır ve altyapı ilgili modülün CffToForm fonksiyonunu tetikler. Sonunda kullanıcı tıkladığı belgeyi açılmış ve değişiklik yapmasına hazır bir şekilde karşısında görür.

2.8. İş Uygulamalarında Bütünleşik Hata ve Log Yönetimi

Yüzlerce kullanıcının aynı veritabanı üzerinde paralel olarak çalıştığı, iş uygulamalarındaki problemlerin önemi nedeniyle Anadil, müşteri, iş ortağı ve geliştirici firma destek ekiplerinin kullanımı için, bağımsız olarak çalışan, fonksiyonel arayüze sahip, Terminal adında bir araç ortaya koymuştur. İş uygulaması çalıştığı sürece, altyapı kütüphanelerinden, veritabanı yönetim sisteminden, iş uygulaması modül, rapor ve servislerinden gelen mesajlar loglanır.

Bilgi, uyarı, hata ve içsel olarak dört mesaj tipi bulunmaktadır. Terminalde biriken mesajlar paketlenip elektronik posta ile incelenmek üzere, uzak bir noktaya yollanabilir. Geliştirici ekip, iş uygulamalarını geliştirirken debug aracı olarak kullanırlar. Kullanıcı tarafında problem yaşandığında ise, destek ekibi uzaktaki kullanıcının Terminalini kendi ekranında izlemeye başlayabilir.



Şekil. 10. Anadil Terminal : iş uygulaması olay izleme aracı.

Terminal, sistemden sisteme ve domainden domaine yönlendirilebilir. İzleme sırasında, kullanıcıya ilgili işlemi tekrarlaması söylenmekte ve hatanın nereden kaynaklandığı bulunmaya çalışılmaktadır.

Hata yönetimi yine altyapıların zayıf kaldıkları katmanlardan biridir. Hatta bir çok geliştirme projesinde hata yönetimi iş akışları seviyesinde ele alınmaktadır. Halbuki iş akışları hata kaynaklarından sadece birisidir. İşletim sistemi seviyesi, ağ problemleri, altyapı kütüphaneleri, veritabanı sunucusu ve iş uygulaması katmanları, program hatası, kullanıcı hatası ve uyarlama parametreleri tanım hatalarını hata kaynakları olarak sayabiliriz. Kullanıcı sayısı ne olursa olsun, veritabanı tutarlılığı, uygulamanın sağlamlılığı ve güvenilirliği, tüm iş uygulamalarında aynı öneme sahiptir. Bu nedenle altyapılar, oluşan hataları yönetebilmeli, her zaman veri tutarlılığını garanti etmeli ve kullanıcıya bir aksiyon alma şansı verebilmelidir. Altyapının kendisi tüm katmanlarda meydana gelebilecek aksiliklere karşı duyarlı ve korunaklıdır.

Veritabanı sunucularından ilgili transaction kapsamında gelen tüm hataları toplar ve Terminale yazar. Transactionların commit ve rollback mekanizmalarını kendisi koordine eder. İlgili belgenin saklanması sırasında bir hata olduğunda, kullanıcıya anlaşılır bir hata mesajı verilir, Terminale tüm detayı ile oluşan hatalar listelenir ve saklanamamış belge kullanıcının karşısında bekler. Destek ekibi gerekli müdahaleyi yapıp problemi giderirse, kullanıcıya sakla tuşuna tekrar basması söylenecektir.

3. Diğer Çerçevelere Göre Konumu

Java EE [3], [4], Oracle Business Components [5], Struts [6], Spring [7] ve .Net [4] platformları ile farklı boyutlarda benzerlikler taşısa da, Anadil Çerçevesi, iş uygulamaları geliştirme ve çalıştırma platformu olarak, bütünleşik bir mimari üzerinde daha dar bir odağa sahiptir.

Tablo. 1. Anadil, Java, Struts ve Spring çerçeveleri karşılaştırma tablosu.

Rich Client	**Anadil**	Evet
	Java	Evet, ancak AWT ve Swing gibi kullanımı kolay olmayan paketler ile yapılıyor
	Struts, Spring	Hayır
Web Application	**Anadil**	Hayır
	Java	Java EE Servlet kütüphanesi ile mümkün. Kolay kullanım için çerçeve kullanımı gerektiriyor.
	Struts, Spring	Evet
Kurulum ve Güncelleme	**Anadil**	Çerçeve içindeki hazır araçlar ile otomatik olarak yapılıyor.
	Java	Java WebStart kütüphanesi ile mümkün.
	Struts, Spring	Uygulamanın Jar şeklinde paketlenerek kopyalanması ile mümkün
Anlık Güncelleme	**Anadil**	Evet. Bileşenlerde yapılan değişiklikler anında yansıyor
	Java	ClassLoader ile uğraşmak gerekiyor
	Struts, Spring	Evet
Web Servisi Erişimi	**Anadil**	Var
	Java, Struts	Java EE kütüphaneleri ile mümkün
	Spring	Var
Güvenlik	**Anadil**	Var
	Java	Kütüphaneler var, hazır yapı yok.
	Struts	Var
	Spring	Yok
Model İzleme Kontrolü	**Anadil**	Evet
	Java	Hayır
	Struts, Spring	Evet
Raporlama	**Anadil**	Var
	Diğerleri	Yok
Geliştirme Ortamı	**Anadil**	Var
	Java	Standart geliştirme ortamı komut satırı araçlarıyla yapılıyor.
	Struts, Spring	Yok. Ancak IDE'ler için geliştirmeyi kolaylaştırıcı eklentiler mevcut
Kolay Veritabanı Erişimi	**Anadil**	Var
	Java	Yok. JDBC kütüphanesi mevcut. Ancak onu kolaylaştırmak için Hibernate gibi ayrı çerçeveler gerekiyor.
	Struts, Spring	Yok. JDBC veya Hibernate gibi başka paketler ile sağlanabiliyor
Otomatik Transaction Yönetimi	Anadil, Spring	Var
	Java, Struts	Yok

Bu sayede, listelenen altyapılardan farklı olarak, zengin arayüz arabirimi, veritabanı erişim yönetimi gibi bir çok katmanı, başka altyapılara yönlendirmeden, hazır ve bütünleşik bir şekilde geliştirici ekiplere ve kullanıcılara sunabilmiştir.

Bununla birlikte sadece Windows platformunda çalışıyor olması, tanıtım ve yaygınlaştırma faaliyetlerinin sınırlı olması nedeniyle bilinirliğinin düşüklüğü, diğer çerçevelere göre zayıf yönleridir.

4. Sonuç

Anadil Çerçevesi'nin doküman merkezcil ve Model Güdümlü mimarisi üzerinde, FMCG (Pepsi Türkiye), Kimya (Jotun Boya), Demir Çelik (Kürüm Holding), Tekstil (Persan Tekstil), Gıda (Kılıçlar Gıda) ve Perakende (Arçelik-Beko bayi entegrasyonu) sektörlerine dönük kapsamlı ERP uygulamaları geliştirilmiştir. İş uygulamaları geliştirme süreçleri altyapının yetkinliğini daha da arttırmıştır. Çözüm ortaklarının da dikey sektörlerde kendilerine ait ürünler geliştirmeye başlamaları ile, bileşen bazlı iş uygulamaları ekosistemi oluşmaya başlamıştır. Tübitak Teydeb tarafından desteklenmekte olan, ARGE projesi kapsamındaki çalışmalar sonucunda, Anadil Çerçevesi, mevcut iş akışlarını koruyarak, üç seviyeli mimariye dönüşümünü tamamlamıştır. Bu mimari dönüşüm ile birlikte, Anadil fonksiyonlarını, web servisi olarak dışarıya açan, veritabanına ve güvenlik sunucusuna yakın konumda, iş akışı bileşenlerini yöneten bir uygulama sunucusu ortaya çıkmıştır. Projenin bir sonraki fazında da, çerçevedeki mevcut prensipler doğrultusunda, ince istemci ortamı için, belge bazlı kullanıcı arabirimin, uygulama sunucusu ve güvenlik sunucusu katmanlarının desteği ile yine sağlam ve kapsamlı iş uygulamaları geliştirme platformunun ortaya konması hedeflenmiştir. Anadil Çerçevesi model güdümlü, açık kaynak kodlu iş uygulamaları zemininde, yazılım ekiplerine başarılı bir mimari sunmaktadır.

5. Referanslar

[1] Object Management Group. http://www.omg.org/mda/ OMG Model Driven Architecture

[2] Stephen J. Mellor., Agile MDA. Project Technology.

[3] http://java.sun.com/j2ee/1.4/docs/tutorial-update2/doc/index.html

[4] http://www.mbi.com.tr/Turkish/Files/ALWSRV_LiteraturAnaliz_Rev_34.pdf

[5] http://www.oracle.com/technology/products/jdev/info/techwp20/wp.html

[6] http://struts.apache.org/1.x/userGuide/index.html

[7] http://static.springframework.org/spring/docs/1.1.5/spring-reference.pdf

GRAFİKSEL KULLANICI ARAYÜZÜ UYGULAMALARI İÇİN TEST EDİLEBİLİR MİMARİ ÖNERİSİ

Ali ÖZZEYBEK
ASELSAN A.Ş.
ozzeybek@mst.aselsan.com.tr

Öz

Elektronik Harp Sistemleri, kullanıcı arayüzünü sağlayan Grafiksel Kullanıcı Arayüzü (GKA) yazılımları, RF sinyalleri toplamaya ve işlemeye yönelik anten ve almaç birimleri, sinyal işleme yazılımları gibi pek çok yazılım ve donanım biriminin bir arada var olduğu karmaşık sistemlerdir. GKA yazılımları ise sistemin işleyişini operatörün yönlendirmesini sağlayan, sistemdeki genel kullanım senaryosunu yöneten ve sistem tarafından bulunan verileri operatöre sunan yazılımlardır.

GKA yazılımlarının artan yetenekleriyle birlikte büyümesi ve dolayısıyla karmaşıklaşması kaçınılmaz olmuştur. GKA yazılımlarının otomatik senaryo testlerinin olması ilerleyen zamanlarda ortaya çıkacak gerek değişikliği, hata düzeltme ve bakım maliyetlerini oldukça azaltacaktır.

Bildiride GKA yazılımları için katmanlı bir mimari önerilmekte ve GKA yazılımlarının otomatik test edilebilmesi için mimari yöntem sunulmaktadır.

1. Giriş

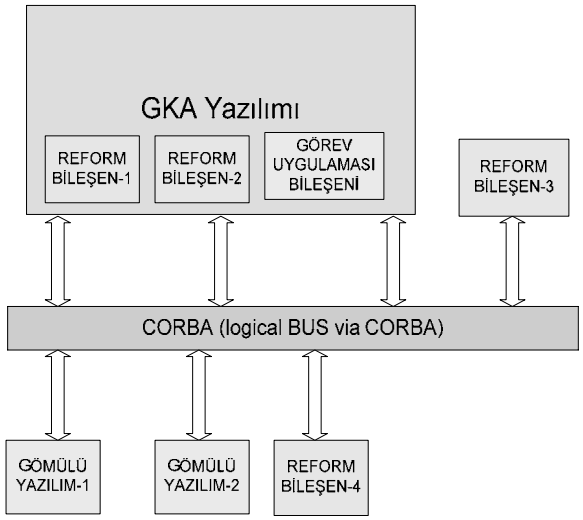
Elektronik Harp Sistemlerinde bütün yazılımların aynı işlemci ve aynı işletim sistemi üzerinde yazılması olası değildir. Bu da dağıtık bir uygulama mimarisini gündeme getirmektedir. CORBA [3] platformdan bağımsızlık, dilden bağımsızlık, performans ve standart haline gelmiş servisleri nedeniyle Elektronik Harp Sistemlerinde sıkça kullanılan bir arakatman teknolojisi durumuna gelmiştir. Bildiride ilk olarak, GKA yazılımının diğer yazılımlarla haberleşme ilişkisini göstermek için CORBA mimarisindeki yerinden bahsedilecektir. GKA yazılımlarını haberleştiği bileşenlerden izole bir şekilde test edebilmek için sahte haberleşme nesnelerinin kullanımı önerilmektedir.

GKA yazılımlarının otomatik test edilebilirliğini zorlaştıran kısım daha çok kullanıcı arayüzü kısımları olmaktadır. Günümüzde GKA yazılımlarını otomatik olarak test eden araçlar bulunmaktadır. Ancak bu araçlarla görsel bileşenlerin, özellikle ekranda gösterilen dinamik grafiklerin test edilmesi oldukça zor olabilmektedir. Bildiride, görsel bileşenleri(panel, buton gibi) otomatik testlere katmadan, ancak bütün senaryonun test edilebilmesini sağlayan bir yöntem sunulacaktır. Görsel bileşenlerin otomatik testlere katılmamasından dolayı burada bahsedilen yöntemin GKA uygulamasını tümüyle test etmediği düşünülebilir. Ancak önerilen tasarım kalıpları ile birlikte görsel bileşenlerin görsel mantık ve iş mantığı içermemesi sağlanmaktadır. Bu da iş mantığını ve görsel mantığı yöneten kısımların otomatik testlerle test edilmesini sağlamaktadır. Bildiride önerilen bu tasarım kalıplarının katmanlı mimarideki yerinden de bahsedilmektedir.

Mimari gösterimler için Unified Modeling Language (UML) kullanılmıştır. Yaratılan “Mimari” ve “Katman” “stereotype”’ları ile UML’in genişletilerek gösterime destek vermesi yöntemi kullanılmıştır.

2. Dağıtık Uygulama Mimarisi

Performans gerekleri nedeniyle farklı işlemciler, farklı işletim sistemleri üzerinde, farklı dillerde yazılan sistem yazılımları görevlerini yerine getirmek için haberleşme mekanizmalarına ihtiyaç duyarlar. GKA yazılımları hem gömülü yazılımlarla hem de REFoRM [7] bileşenleriyle haberleşebilmek için bir arakatman teknolojisi kullanmak durumundadır. Arakatman teknolojisi olarak bu tip İstemci-Sunucu mimarilerinde kendini ispatlamış bir teknoloji olan CORBA tercih edilmiştir. Şekil-1’de GKA yazılımının gömülü yazılımlarla ve REFoRM bileşenleriyle ilişkileri CORBA referans mimarisi üzerinde gösterilmektedir [3].



Şekil – 1 Dağıtık Uygulama Mimarisinde GKA Yazılımının Yeri

Şekil – 1'de görüldüğü gibi GKA Yazılımı bazı REFoRM bileşenlerini CORBA servisi gibi kullanırken bazı bileşenleri de kütüphane olarak kullanabilmektedir.

3. GKA Yazılım Mimarisi

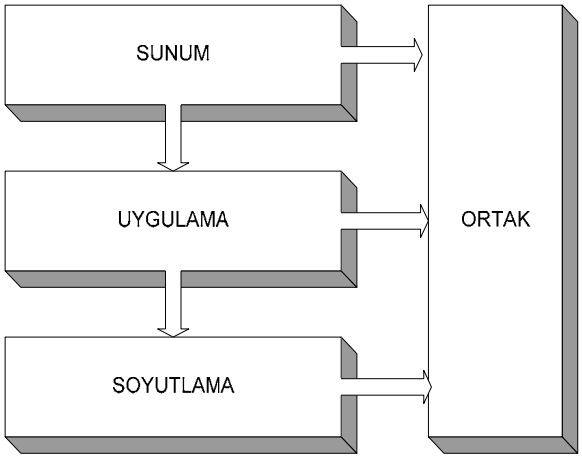
GKA yazılımlarında modülerlik ve bakım yapılabilirlik gibi mimari kalite gereklerinden dolayı katmanlı tasarım kalıbı yaygın olarak kullanılmaktadır. Bu kısımda katmanlı tasarım kalıbını kullanmamıza neden olan gereklerden ve mimarideki katmanlardan bahsedilecektir.

GKA yazılımlarında gerek değişiklikleri en çok görünümle ilgili olmaktadır. Önerilen mimaride görünüm değişikliklerinin yazılımın diğer kısımlarını etkilemeden yapılması istenmektedir.

Geliştirilen farklı GKA yazılımlarında birbiriyle aynı işi yapan fakat farklı yazılmış kod parçaları olduğu gözlenmiştir. Bu kod parçaları diğer projelerde kullanılmak istendiğinde bağımlılıkların buna izin vermediği görülmüştür. Önerilen mimarinin bu tip kütüphane olarak kullanılabilecek paketlerin tekrar kullanılmasını sağlayacak önlemleri alması beklenmektedir.

Ayrıca GKA yazılım mimarisinin otomatik senaryo testlerinin gerçekleştirilebilmesi için gerekli önlemleri alması beklenmektedir.

GKA yazılımında modülerlik, anlaşılabilirlik, test edilebilirlik ve tekrar kullanılabilirlik gereklerinden dolayı katmanlı mimari tasarım kalıbı [1] kullanılmıştır. Katmanlar soyutlama seviyesine göre seçilmiştir. Şekil-2'deki soyutlama katmanı uygulama katmanını diğer yazılım ve donanım arayüzlerinden soyutlarken, uygulama katmanı sunum katmanını iş senaryolarından soyutlamaktadır.



Şekil – 2 GKA Yazılımı Katmanlı Mimarisi

Katmanların detaylarına gelince; en alttaki katman Soyutlama Katmanı'dır. Soyutlama Katmanı'nın görevi GKA yazılımını diğer yazılım ve donanımlardan soyutlamaktır. Bu katmandaki paketlerin birbirlerine bağımlılığı olmayacaktır. Bu şekilde bu katmandaki paketlerin taşınabilirliği de sağlanabilmektedir. Ancak taşınabilirlik için yönetimsel faaliyetlerin de olması gerektiğini unutmamak gerekir.

Ortadaki katman Uygulama Katmanı'dır. Bu katman iş mantığının, iş alanı ile ilgili paket ve sınıfların yer aldığı katmandır. Uygulamanın senaryolarını yöneten iş modelleri bu katmanda yer almaktadır. Bu katmanda yapılan işin gösterimiyle veya haberleşmeyle ilgili paketler yer almaz.

Sunum katmanı ise görünümle ilgili paket ve sınıfların yer aldığı, pencere gösterim ve yönetimlerinin yapıldığı en üstteki katmandır.

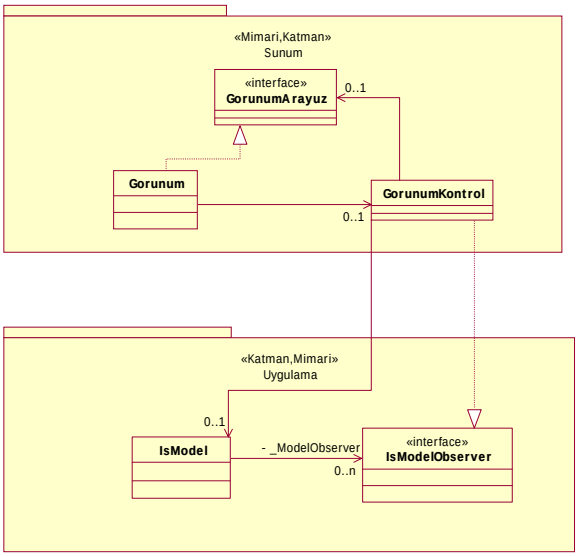
Ortak pakette ise KA yazılımları arasında ortak kullanılabilecek paketler yer almaktadır. Ortak pakette yer alan paketlerin birbirlerine bağımlılıkları olmamalıdır. Bu önlem buradaki paketlerin taşınabilirliğini sağlamak için alınmıştır.

Bu mimaride bağımlılık yönü aşağıya doğrudur. Yani Uygulama Katmanı, Sunum Katmanı'nı, Soyutlama Katmanı ise Uygulama ve Sunum Katmanlarını bilmez. Bu şekilde KA yazılımında iş mantığını etkilemeyen görsellikle ilgili değişikliklerin sadece Sunum Katmanı'nda kalması sağlanmaktadır.

4. Mimaride Kullanılan Tasarım Kalıpları

Sunum ve Uygulama katmanlarındaki işlevleri ayırmak için Model-View-Controller (MVC) ve Model-View-Presenter (MVP) tasarım kalıpları kullanılmıştır [6]. Bu tasarım kalıpları görünümle ilgili sınıfların yükünü azaltması sayesinde otomatik müşteri testlerinin ve birim testlerin yazılmasını kolaylaştırmıştır [4].

MVC ve MVP tasarım kalıplarında geçen iş modelleri UygulamaKatmanı'nda yer almaktadır. MVC'deki Kontrol sınıfı (Controller) ve MVP'deki Hazırlayıcı sınıfı (Presenter) tek bir sınıfta toplanmış ve Şekil – 3'de GorunumKontrol ismini almıştır. Buradaki GorunumKontrol sınıfının görevi Mediator [5] tasarım kalıbına uymaktadır.

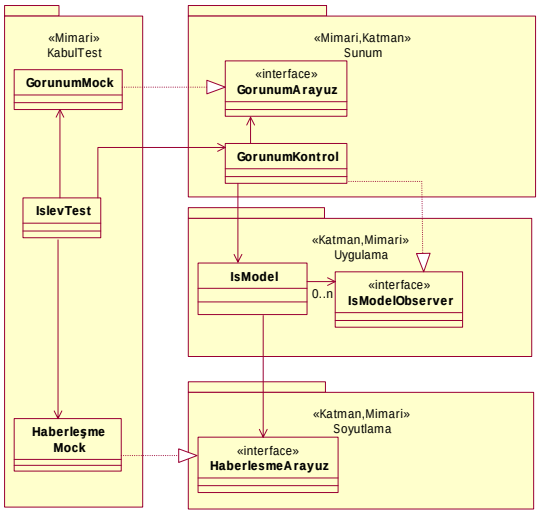


Şekil – 3 Katmanlar arası MVC-MVP Görünümü

Görünümle ilgili değişiklikleri GorunumKontrol sınıfı IsModelObserver sınıfından aldığı uyarı ile gerçekleştirmektedir. Bu şekilde görünümle değişikliklerin kullanıcı arayüzü tetiklediği anda değil iş modellerinde değişiklik gerçekleştiği anda yapılması sağlanmıştır. Bu yönteme Abone Senkronizasyonu denilmektedir.[2]

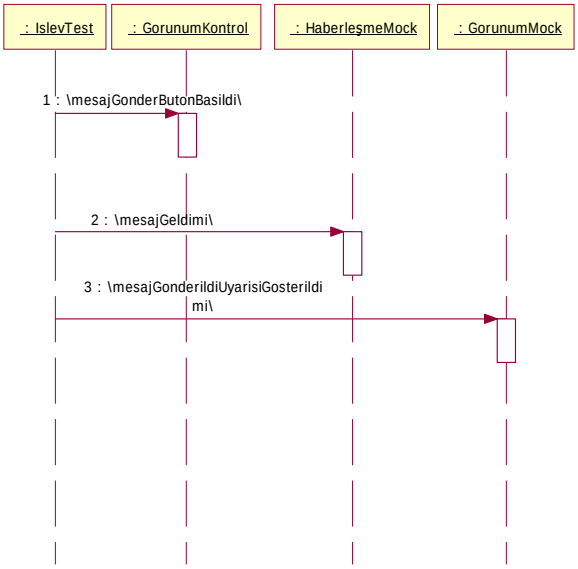
5. Otomatik Kabul Testlerinin Mimarideki Yeri

KA yazılımlarının otomatik müşteri testlerini ve sunum katmanının birim testlerini yazmak görünümle ilgili sınıflar nedeniyle zordur. Bu zorluk görünümle ilgili sınıfların çokça değişmesinden ve hazır görünüm kütüphaneleri (Java Swing gibi) kullanılmasından kaynaklanmaktadır. Şekil – 4'de bir işlev test edilirken hangi katmanların kullanılacağı ve katmanlar arası ilişkiler gösterilmektedir.



Şekil – 4 GKA Yazılımı Test Görünümü

Müşteri test senaryoları GorunumKontrol sınıflarından başlamaktadır. (IslevTest sınıfının GorunumKontrol sınıfındaki IslevButonunaBasildi metodunu çağırması gibi) IslevTest sınıfı daha sonraki detaylarla ilgilenmemektedir. Test sınıfı GKA yazılımının haberleştiği birimlerin sahtelerini kullanarak haberleşme senaryosunu test etmekte, görünüm sınıflarının sahtelerini kullanarak da görünümle ilgili değişiklikleri test etmektedir. Şekil – 5'de GKA yazılımının bir CORBA bileşenine mesaj gönderme senaryosu anlatılmaktadır. Test Kontrol sınıfından başlamakta, sonuçları HaberlesmeMock ve GorunumMock sınıflarından toplanmaktadır.



Şekil – 5 Örnek Mesaj Gönderme Senaryosu

Burada müşteri testlerinde gerçek görünüm sınıfları (Panel, dialog gibi) kullanılmadığı için test töntemi yazılımın tümünü kapsamamaktadır. Görünüm sınıflarındaki işlevlerle ilgili kodun sadece görünümle ilgili olmasını ve 1-2 satırı geçmemesini ve buradaki görünüm mantığı ile ilgili işin GörünümKontrol sınıfı aracılığıyla yapılmasını sağlayarak görünüm mantığının da testlerde kapsanması sağlanmaktadır.

Haberleşmeler CORBA üzerinden yapıldığı için arayüz sınıfı üzerinden sahtesinin yaratılması çok kolay olmaktadır. Çünkü Interface Description Language (IDL)'le yaratılan arayüz kodlarının gerçeklenmesi tamamen nesne yönelimli mantık ile yapılmakta hiç haberleşme detayı içermemektedir. Böylece daha alt seviyede gerçekleşebilecek hatalarla ilgili endişe duyulmamaktadır.

6. Sonuç

Bu bildiride Elektronik Harp projelerinde GKA yazılımlarının otomatik testlerinin yapılabilmesi için bir GKA mimarisi önerilmiştir. Önerilen mimari kullanıma alınmış ve uygulanabilirliği ispatlanmıştır. Otomatik müşteri testleri için FIT [8] altyapısı kullanılmıştır.

GKA yazılımında uygulanan katmanlı mimari tasarım kalıbı yazılımın modülerliğine önemli ölçüde katkıda bulunmuştur. Katmanlı tasarım ile sağlanan modülerlik dikeyde de MVC ve MVP tasarım kalıplarıyla desteklenmiştir. Ancak Soyutlama katmanına gelen verinin Sunum katmanında gösterilmesi ihtiyacı katmanlı tasarım kalıbında veri tekrarlarına dolayısıyla da performans kayıplarına yol açmıştır.

Abone Senkronizasyonu yöntemi ile görünüm bileşenlerinin iş modelleri ile senkronizasyonu sağlanmıştır. Bu yöntem ile karmaşık senkronizasyon problemlerinin üstesinden gelinmiştir. Ancak katmanlar arasındaki yüklü abone ilişkisi yazılımın çalışma zamanı ilişkilerinin karmaşıklaşmasına yol açmıştır.

GKA yazılımlarında sorun olan otomatik müşteri testleri sahte sınıfların yaratılması ile çözülmüştür. Otomatik müşteri testleri sırasında gerçek görünüm bileşenleri yaratılmamış onların yerine sahteleri geçmiştir. Gerçek görünüm bileşenlerinin testte olmaması test mantığı olarak risk teşkil etse de bunun kabul edilebilir bir risk olduğu söylenebilir. Çünkü MVP tasarım kalıbındaki hazırlayıcı sınıflar gerçek görünüm bileşeninin sunması gereken bilgiyi hazırlayarak görünüm bileşeninin hiç akıl taşımamasını sağlamıştır[4].

7. Kaynaklar

[1] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley, 1996

[2] Martin Fowler, *GUI Architectures*, www.martinfowler.com, 2006

[3] OMG, *Common Object Request Broker: Architecture and Specification*, 1998.

[4] Feathers, Michael, *The Humble Dialog Box*, mfeathers@object.com

[5] Gamma, et. al., *Design Patterns*, Addison Wesley, 1995

[6]Martin Fowler, *Model View Presenter*, www.martinfowler.com/eaaDev/ModelViewPresenter.html, 2004.

[7] ASELSAN MST Grubu, *REFoRM – RADAR Elektronik Harp Fonksiyonel Referans Modeli*, 2004

[8] W. Cunnigham, FIT: Framework for Integrated Test, http://fit.c2.com, 2005

Modelleme

Representing Feature Models with Semantic Web Ontologies

Veli Biçer[1], Cengiz Toğay[2]

[1]*R & D Department*
Tepe Technology
Ankara, Turkey

[2]*Department of Computer Engineering*
Middle East Technical University
Ankara, Turkey

[1]vbicer@tepeteknoloji.com.tr
[2]ctogay@gmail.com

Abstract

Feature models play a key role in the domain analysis process of the software product lines. The common and variable features captured in the domain analysis guide the further domain engineering steps by providing the system scope, and constituting a basis for the system architecture, reusable components and the domain-specific languages. In order to fully benefit from the feature models in the generative software platform, a formal methodology and semantics is required. In this paper, we propose a three-layered approach to define the feature models by using the Semantic Web technologies such as Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL).

1. Introduction

Feature models are extensively used for domain analysis within a product line development process[8][6] and locating components in component-oriented software engineering approaches [15][16][17]. The commonalities and variabilities captured as a result of the feature modeling process is not only practical to define the scope of the product lines, but it is also invaluable in further steps as it provides a basis for defining system components, common architecture and configuration methodology.

On the other hand, the idea of generative software development [4] aims at automating software product line development. This is achieved by utilizing the systematic software reuse, creating product-line elements such as models, services, generators, domain-specific languages and using some static and dynamic technologies such as metadata, reflection, configuration or aspect-oriented techniques.

Feature models play a key role in generative software development [4]. Firstly, feature-oriented domain analysis let us to define a system family in detail. More importantly, the feature model created as a result of domain analysis provides the foundations of the domain-specific languages (DSLs). In addition, they are the crucial factor in the transition from the problem space to the solution space. The common architectures and services can be derived based on the feature models which are later reused intensively in the product development process.

In order to utilize a generative environment for a product-line, there is a need for the formal and machine-processable feature models. This is due to the fact that the feature models are needed to be used in defining architectures, generating codes, and developing product automatically as they provide the base for all those activities. In this work, we propose a methodology and representation mechanism by using the Semantic Web technologies such as Web Ontology Language (OWL), and Semantic Web Rule Language (SWRL) in order to fully benefit from the feature models in all the steps of the product-line and product development.

2. Related Work

2.1. Feature Oriented Software Engineering

Feature modeling is a key approach to define and analyze the commonalities and variabilities for a particular product family as well as specifying the dependencies among them [8]. The main entity of the *feature* modeling is the feature through which the well-known and individual aspect of the software system is specified. The features systematically constitute a domain analysis model which is mostly represented by one or more feature diagrams.

The domain analysis model is, then, used in the further steps such as the domain design, implementation and the application development in order to define and use the reusable core assets in a feasible and hierarchical way. In addition, this model provides the basis for the domain which may lead to other elements such as domain-specific languages, frameworks, and common architectures.

Features within a feature diagram can be stated as mandatory, optional, alternative and or-relation. A feature is mandatory if each product in the family has it as an aspect. For example, all the cars have an engine although the types of the engine may vary. On the other hand, a feature is optional if its inclusion in the system depends on the particular product instance. In addition, a set of features are called alternatives if exactly one feature can be selected among them. As an example, for a car, only one transmission can be selected among the manual and automatic transmission types. Including one or more features from a set of features are also possible through the "Or" relationship among these features. The graphical notation of these feature types are shown in Figure 1.

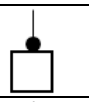
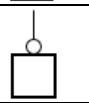
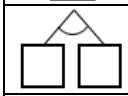
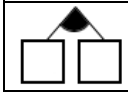
	Mandatory Feature
	Optional Feature
	Alternative Feature
	Or-relation Feature

Figure 1 Feature notations

The core feature model approach, presented in [8], has been expanded with the introduction of the new extensions and variations in the recent years [2][3]. Cardinality based feature models [2] annotates the features with the cardinalities in order to specify how often the features are present in the feature configuration. In this approach, there are three kinds of features which are the root feature, the solitary feature and the grouped feature. A root feature represents the root concept in the diagram. A grouped feature is, on the other hand, the one that occurs in a feature group. A solitary feature, as the name implies, is a single feature which is not included in a feature group. In cardinality-based feature models, every solitary feature is annotated with the feature cardinality to specify the number of times the feature is included in the configuration. This is expressed with the intervals of the form [m..n] where m and n represent the minimum and maximum values of the interval, respectively. For example, the mandatory feature and the optional feature, shown in Figure 1, is represented as the solitary features with the intervals [1..1] and [0..1] in the cardinality, respectively. In addition, the features can be organized into feature groups where for each group a group cardinality is specified. This is actually an interval of the form <m..n> in order to state how many features can be selected from the group. As an example, an alternative feature, shown in Figure 1, is denoted as a feature group with the cardinality <1..1>.

In addition to the cardinalities, the feature diagram references can be used for the modularization of the feature models [3]. A feature diagram reference is a node in the feature model referring to a separate feature diagram. This enables dividing the large feature diagrams into smaller ones as well as introducing the reusability.

Features may also be associated with a type in order to specify the attributes in the feature models[5]. Attributes are crucial in feature models as it introduces the parameterization at the domain analysis level by providing a way to represent the values of a domain. In addition, a feature may also contain a set of attributes modeled as subfeatures associated with the desired types.

As well as these extensions, different relationships among the features and categorization mechanisms are introduced in [10][14][12]. In this work, however, we consider the core feature model approach, cardinality extensions, modularization and attributes to be represented by Semantic Web Ontologies for the brevity. However, it is possible to reflect any kind of extensions to the ontologies due to the extensible nature of the Semantic Web.

2.2. Generative Programming

Generative programming [4] aims at automating application development by using the generators and the reusable software components. In order to enable to automation, the generative programming focuses on

the domain engineering cycle to create the generators, reusable components, and common architectures.

Generative programming uses the feature oriented approach for domain analysis with the aim of finding common and variable features of the applications. The models obtained as a result of the domain analysis provides the structural and behavioral aspects of the domain concepts.

Based on the results of the feature models, the development of a common architecture for an application family is specified. The domain implementation step involves creating reusable components, generators and the domain-specific languages.

Generative programming approach achieves the automation of the application development through a specification written in the domain-specific languages. Domain specific languages offer the expressiveness on a particular domain with the help of the abstractions, concrete syntax, and domain-specific tools. In order to create domain-specific languages, the information through the domain engineering life cycle is clearly captured and represented with its formal semantics. Semantic Web can play a key role for the automation of the application engineering with the help of the ontologies, rule languages, and its standardized, and extensible nature.

2.3. Web Ontology Language (OWL)

Web Ontology Language (OWL) [13] is a semantic markup language developed by the World Wide Web consortium for publishing and sharing ontologies. OWL is based on the Resource Description Framework (RDF) [9]. The relation between RDF and OWL is depicted in Figure 2.

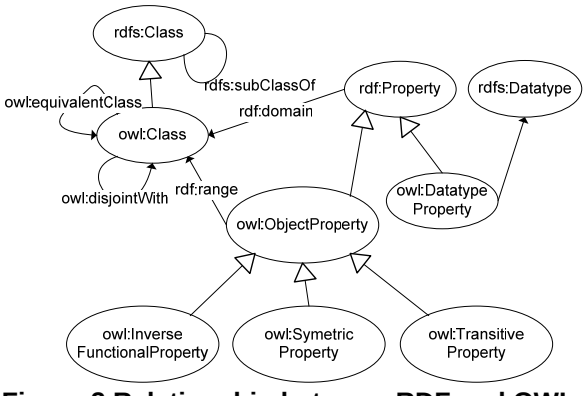


Figure 2 Relationship between RDF and OWL

OWL describes the structure of a domain in terms of classes and properties. Classes can be names (URIs) or expressions. Furthermore, the following set of constructors are provided for building class expressions: owl:intersectionOf, owl:unionOf, owl:complementOf, owl:oneOf, owl:allValuesFrom, owl:someValuesFrom, owl:hasValue.

In OWL, properties can have multiple domains and multiple ranges. Multiple domain (range) expressions restrict the domain (range) of a property to the intersection of the class expressions.

Another aspect of the language is the axioms supported. These axioms make it possible to assert subsumption or equivalence with respect to classes or properties. The following are the set of OWL axioms: rdfs:subClassOf, owl:sameClassAs, rdfs:subPropertyOf, owl:samePropertyAs, owl:disjointWith, owl:sameIndividualAs, owl:di_erentIndividualFrom, owl:inverseOf, owl:transitiveProperty, owl:functionalProperty, owl:inverseFunctionalProperty.

2.4. Semantic Web Rule Language (SWRL)

Semantic Web Rule Language (SWRL) [7] is an extension of OWL semantics to include the rules. It is mainly based on a combination of the OWL DL and OWL Lite with the Unary/Binary Datalog RuleML sublanguage of the Rule Markup Language. The rules specified in SWRL are of the form of an implication between the rule body (antecedent) and the rule head (consequent). The rule in this form indicates that if the body holds, then the head must also hold.

In SWRL, the OWL axioms are extended by introducing the rule axiom. The rule body and head contains atoms to specify the conditions to be hold. Atoms can be of the form C(x), P(x,y), sameAs(x,y), differentFrom(x,y), or builtIn(r,x,y,…). Here, C represents an OWL class or data range, P denotes an OWL property, r is a built-in relation, and x,y are the OWL individuals as variables. C(x) is true if x is an instance of class or data range C. P(x,y) holds if there is a property P which relates x to y. sameAs(x,y) is true if x and y are the same instances whereas differentFrom(x,y) is true if x and y are different. builtIn(r,x,y,…) holds if the relation r, which is specified as one of the built-ins in SWRL[7] such as comparisons, math or string functions, is true for the given arguments.

SWRL provides a powerful mechanism to specify the rules on OWL. As well as its abstract syntax, it is possible to specify the SWRL rules in its XML concrete syntax. For example, the following *inverseOf* property rule of OWL can be stated in SWRL as shown in Figure 3:

- P1(x,y)→ P2(y,x)

```
<ruleml:imp>
 <ruleml:_body>
  <swrlx:individualPropertyAtom
   swrlx:property="P1">
   <ruleml:var>x</ruleml:var>
   <ruleml:var>y</ruleml:var>
  </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
   <swrlx:individualPropertyAtom
    swrlx:property="P2">
     <ruleml:var>y</ruleml:var>
     <ruleml:var>x</ruleml:var>
   </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

Figure 3 SWRL XML concrete syntax

3. Representing Feature Models in OWL

The use of feature models to capture the common and variable features of the systems within a particular domain leads to easily determining the scope of the applications in a software product line. In addition to this, feature models provide input to the other stages of the domain engineering process such as constructing a common architecture model or creating a domain-specific language. Therefore, the representation of the feature models in a standardized and machine-processable way is one of the initial steps for an automated product-line construction.

In this section, we show how to represent the feature models in OWL to provide a formal mechanism to domain analysis. Actually, the basics of feature model representation in OWL are previously introduced in [18]. In this work, we propose a new approach for the representation of the feature models based on the semantics of the recent extensions to the feature modeling introduced in [2][3][5]. We also utilize the rules with the SWRL on the information represented with the ontologies.

A three layered approach is proposed for representing the feature models in OWL. This is depicted in Figure 4. The first layer consists of the feature model meta-ontology through which the meta-data of the feature modeling process is defined. In fact, this ontology involves the generic classes and the relationships to guide the feature modeling process. The second layer includes the actual feature model as classes and their relationships which are derived based on the meta-ontology of the first layer. In the third layer, the instances of a particular feature model are created as a result of feature configuration process. The first and the second layers of our approach enable managing the feature modeling process for the product-line development. On the other hand, the third layer is mainly used for a specific product development.

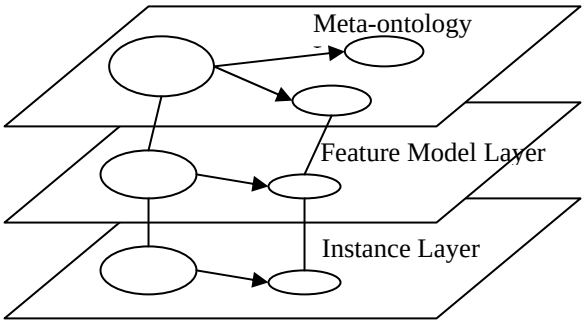


Figure 4 Three layers of the feature models representation in OWL

The benefits of this layered approach can be stated as follows:

- The availability of a generic meta-ontology for feature modeling coordinates the modeling process through the well-established basis and formal mechanisms
- Feature model ontology also provides the formal semantics of the product-line scope which is then realized through the instances at the product development stage.
- The ontologies of the first and second layer enable the use of inference and reasoning capabilities at class-level in the product-line development, whereas the instance-level reasoning is also introduced in the third layer.

In order to model the features of an application family, the feature modeling meta-ontology in the first layer provides the basic concepts and relationships. To represent each node in a feature model, we create a ModelNode class. The ModelNode class is actually an abstraction to represent every kind of nodes to be presented in a feature model. There are three kinds of specialization for the ModelNode class in the meta-ontology which are the RootNode, FeatureNode, and GroupNode to represent the root features, solitary features and the feature groups as proposed in cardinality-based feature models [2]. In addition to these, any possible extension to the nodes in the feature models can also be included as a subclass of the ModelNode class. This is illustrated in Figure 5 in OWL abstract syntax[11].

```
Class(ModelNode partial)

Class(FeatureNode partial ModelNode)

Class(RootNode partial ModelNode)

Class(GroupNode partial ModelNode)
```

Figure 5 Class definitions for node representation

In addition to the nodes, a feature model diagram also includes feature relations to represent the associations among the nodes. Although, basically, these edges can be directly mapped to the object properties in OWL, they mostly mean more than just a binary relation. In fact, the edges include some additional properties and semantics such as cardinalities, direction, or constraints. Therefore, we represent these feature relations as OWL classes since an edge in the feature model is rather a concept with its own properties and semantics. Furthermore, at the meta-ontology level, our main aim is to model the feature modeling approach by using the mechanisms and syntax of the OWL. Therefore, it is more convenient to use the OWL axioms as a tool to represent more domain specific abstractions such as features, feature relations and their properties.

In order to represent the feature relations in the feature model, an abstract *ModelLink*, shown in Figure 6, class is created. *ModelLink* is the basis for all the associations among the nodes of the feature model. One of the crucial associations in a feature model is the "subfeature of" link through which the parent-child relation between the features is represented. Therefore, we derive a class from the *ModelLink*, namely *SubFeatureLink*, in our meta-ontology to denote this association. Although the "subfeature of" relation is an association in the feature model, it is a specific entity for our OWL based meta-ontology with its own properties and restrictions. The *SubFeatureLink* class has two important properties.

The *hasSubFeature* property relates the *SubFeatureLink* class to a particular solitary feature which can be of type *FeatureNode* or *GroupNode*. In addition, the *hasSubFeature* can refer to *RootNode* relating the *SubFeatureLink* to another feature diagram. Therefore, the modularization of the feature model is achieved by using the *RootNode* where needed and pointing to it through the link classes.

A *someValuesFrom* restriction is used on the *hasSubFeature* property referring to the union of *FeatureNode* , *GroupNode* and *RootNode*. We also restrict that a SubFeatureLink can point to only one subfeature. The semantics are shown in Figure 6.

```
Class(ModelLink partial)

Class(SubFeatureLink partial ModelLink)

Class(SubFeatureLink partial restriction
(hasSubFeature someValuesFrom
unionOf(FeatureNode GroupNode)))

Class(SubFeatureLink partial restriction
(hasSubFeature cardinality 1))
```

Figure 6 Class definitions for link representation

The cardinality-based feature models also requires the solitary features to be specified with the feature cardinalities in order to indicate the number of times a feature can occur in the configuration. Feature cardinality actually refers to a sequence of intervals in which the minimum and maximum values of the interval is denoted. In order to introduce the feature cardinalities in the meta-ontology, a *FeatureCardinality* class is created which is a subclass of the abstract *Cardinality* class. The *Cardinality* class has a *hasInterval* property whose range points to another class called *Interval* to specify the minimum and maximum values of the cardinality. The *FeatureCardinality* class further restricts the *hasInterval* property with the *someValuesFrom* restriction to state that there is at least one interval for each feature cardinality.

The feature cardinality is essentially specified as a property of the "subfeature of" relationship for each solitary feature in the feature model. Therefore, the other property of the *SubFeatureLink*, namely *hasCardinality*, is used to associate cardinality with the subfeature. The *hasCardinality* property points to the *FeatureCardinality* class and each *SubFeatureLink* has only one *hasCardinality*. The semantics are depicted in Figure 7.

The "subfeature of" relationship specified by the *SubFeatureLink* class is actually a property of the solitary and the root features. However, the *GroupNode* differs from these nodes in terms of the association with its subfeatures. A group node represents a choice over a set of grouped features with the restriction specified through the group cardinality. The group cardinality <n-m> specifies that one has to select at least n and at most m distinct grouped features in the group. In order to represent this association, we create another subclass of the *ModelLink* class, namely *GroupLink*. *GroupLink* points to one or more grouped feature through the *hasGroupedFeature* property which can be of type *FeatureNode* or *RootNode* (for modularization purposes). Unlike the *hasSubFeature*

property of the *SubFeatureLink*, the cardinality of the *hasGroupedFeature* can be more than one since there can be a number of grouped features in the group. Similar to the *SubFeatureLink*, the *GroupLink* has the *hasCardinality* property to specify the group cardinality. Observe that *hasCardinality* points to another subclass of *Cardinality*, namely *GroupCardinality*, since the *GroupCardinality* is specified with only one interval whereas the *FeatureCardinality* may include more than one interval. The details are shown in Figure 8.

```
Class(Cardinality partial)

Class(FeatureCardinality partial
Cardinality)

Class(Interval partial)

Class(SubFeatureLink partial restriction
(hasCardinality someValuesFrom
FeatureCardinality))

Class(SubFeatureLink partial restriction
(hasCardinality cardinality 1))

Class(FeatureCardinality partial
restriction (hasInterval someValuesFrom
Interval))

DatatypeProperty(IntervalMinValue
domain(Interval) Functional)

DatatypeProperty(IntervalMaxValue
domain(Interval) Functional)
```

Figure 7 Feature cardinality semantics

```
Class(GroupLink partial ModelLink)

Class(GroupLink    partial    restriction
(hasGroupedFeature        someValuesFrom
FeatureNode))

Class(GroupCardinality            partial
Cardinality)

Class(GroupLink    partial    restriction
(hasCardinality           someValuesFrom
GroupCardinality))

Class(GroupLink    partial    restriction
(hasCardinality cardinality 1))

Class(GroupCardinality            partial
restriction  (hasInterval  someValuesFrom
Interval))

Class(GroupCardinality            partial
restriction (hasInterval cardinality 1))
```

Figure 8 Group links and cardinalities

As stated in section 2, the features can be associated with the types to create attributes. Then the attributes are assigned with a value at the configuration level. This is the case for the *FeatureNode* in our meta-ontology. Features derived from the *FeatureNode* can be assigned a value at the instance layer. In order to enable this, a datatype property, *hasValue*, is defined for the *FeatureNode*. However, at the meta-ontology level, the range of this property is not specified. This is by assigning an XML Schema data type to the range of the *hasValue* property at the feature model ontology in the second layer.

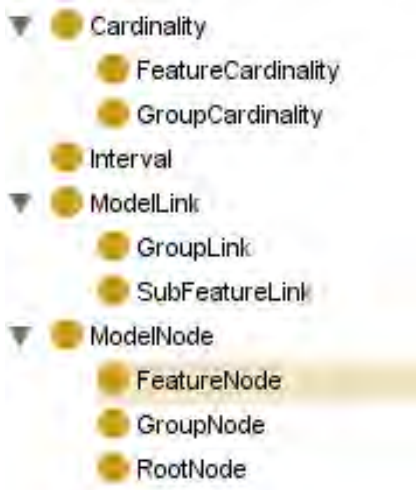


Figure 9 Meta-ontology classes

By using the semantics of the meta-ontology, feature model ontology is constructed in the second layer. Each node in the feature model is defined as a class which inherits from one of the subclasses of the *ModelNode* class of the meta-ontology. For example, each root node is defined as a new class which is a subclass of the *RootNode* class. Similarly, any solitary feature and grouping node are also defined as classes inheriting from the *FeatureNode* and *GroupNode* classes, respectively. Therefore, any node in the feature model automatically inherits all the properties and restrictions of the corresponding node defined the feature model semantics.

In addition, the associations in the feature model are also represented as classes inherited from the *ModelLink* class or its subclasses. A “subfeature of” relationship, for example, is introduced as a class inherited from the *SubFeatureLink* class. Furthermore, a grouping association between the grouping node and the child is represented as a class inherited from the *GroupLink*.

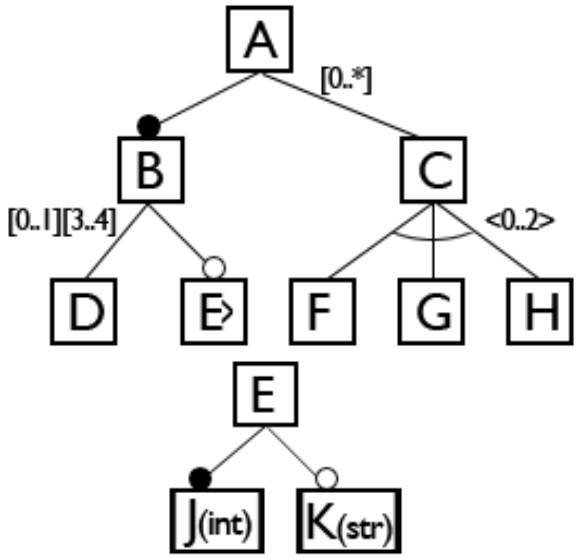


Figure 10 An example feature model

In order to clarify the feature modeling step, consider the feature diagrams shown in Figure 10. In this figure, there are two feature diagrams since we have two root nodes, namely A and E. These nodes are defined in the feature model ontology as subclasses of the *RootNode*. In addition to those, there are solitary features such as B, D, or J which should be included in the ontology as a subclass of *FeatureNode*. Furthermore, the node C is a grouping node. Therefore, it is represented by a class inherited from the *GroupNode*.

Once the nodes are defined, the relations among them can be constructed by using the semantics of the meta-ontology. For example, the relation between the A and B is represented by a class (e.g. named as *AtoB*) which inherits from the *SubFeatureLink* and requires two additional restrictions:

- Given that class *A* is inherited from the *RootNode*, class *B* is inherited from *FeatureNode* and *B* is a subfeature of *A*, we define the class *AtoB* and create a restriction on *A* that *A* has the property *hasSubFeatureLink* which has "some values from" *AtoB*.
- In addition, there is a restriction on *AtoB* that its *hasSubFeature* property has "some values from" *B*.

By using these kinds of class-level restrictions, we can create the all associations in the ontology. Similar approach is also used for the group links. The semantics of the association between A and B are shown in Figure 11.

```
Class(A partial RootNode)

Class(B partial FeatureNode)

Class(AtoB partial SubFeatureLink)

Class(A partial restriction
(hasSubFeatureLink someValuesFrom
AtoB))

Class(AtoB partial restriction
(hasSubFeature someValuesFrom B))
```

Figure 11 Representing Features A and B

The cardinalities associated with the links can also be specified by utilizing the *hasValue* restriction. In the example, *B* is a mandatory feature which can be regarded as a solitary feature with a cardinality [1..1]. Therefore, the "subfeature of" relationship, represented by *AtoB*, need to implicitly (through a subclass of *FeatureCardinality* class according to the meta-ontology) refer to a class, namely *AtoBInt*, inherited from Interval with the following restrictions:

- To specify the maximum value of the interval as one, a restriction on *AtoBInt* is created such that *IntervalMaxValue* property of the *AtoBInt* has value 1.
- Similarly, to specify the minimum value of the interval as one, a restriction on *AtoBInt* is created such that *IntervalMinValue* property of the *AtoBInt* has value 1.

The semantics of the *AtoBInt* and its association to *AtoB* is depicted in Figure 12.

```
Class(AtoBCard partial
FeatureCardinality)

Class(AtoBInt partial Interval)

Class(AtoB partial restriction
(hasCardinality someValuesFrom
AtoBCard))

Class(AtoBCard partial restriction
(hasInterval someValuesFrom AtoBInt))

Class(AtoBInt partial restriction
(IntervalMaxValue hasValue 1))

Class(AtoBInt partial restriction
(IntervalMinValue hasValue 1))
```

Figure 12 Specifying the cardinalities

The feature diagram references in the model are represented by creating a SubFeatureLink from parent feature in the main diagram to the root node of the

target diagram. In Figure 10, a SubFeatureLink from the class B to the class E, inherited from RootNode, is used for diagram referencing.

Furthermore, in Figure 10, the features J and K is annotated with the types. Therefore, the *hasValue* property, defined in meta-ontology, is needed to be restricted to the specific types in order to specify the values in the configuration. This restriction for feature J is shown in Figure 13.

```
Class(J partial FeatureNode)

Class(J partial restriction (hasValue
someValuesFrom xsd:int))
```

Figure 13 Specifying attribute ranges

Once the feature models are specified as ontologies, the instances of the ontologies can be created as feature configuration. However, to guide the configuration process at the instance level, we also need to consider the constraints. Therefore in the next section, the rules are introduced based on the ontologies defined for feature modeling.

4. Representing Constraints as Rules in Semantic Web

In order to fully exploit the power of the Semantic Web for representing the feature models, we need the rules on the top of the feature model ontology. The representation mechanism presented in the previous section provides data about the feature models as a pool of information in OWL. Instead, the rules are the actual control mechanism for the inference steps in order to specify the conditions that should be satisfied in the feature configuration.

By using the SWRL, it is possible to specify the constraints for the feature models at any level as the rules on the OWL ontologies. The most common constraints in the feature model are the *"requires"* and *"excludes"*. The *"requires"* constraint implies that given two features A and B, A *requires* B means that A is a feature dependent on B in order to be functioning. The *"excludes"* constraint, on the other hand, means that a feature configuration of a product cannot involve A and B features at the same time.

For the feature model example shown in Figure 10, assume that there is *requires* constraint between features B and C as depicted in Figure 14. Since the features B and C are represented as OWL classes in the second layer (feature model ontology layer), a SWRL rule to state the *requires* can be expressed as follows:

- B(?x) → C(?y)

This indicates that if an instance(x) of feature B is present in the configuration, an instance(y) of feature C is also included in the configuration.

Assuming that there is an *excludes* constraint between the feature F and G in the example in Figure 10, this constraint can be stated in SWRL as follows:

- F(?x) → ~G(?y)

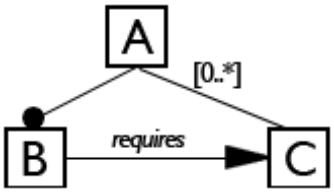


Figure 14 Requires constraint

6. Conclusion and Future Work

Feature modeling is a common approach to define the structural and behavioural aspects of an application family with the commonalities and variabilities. It is also important since its results can used in all steps of the domain engineering process.

In this work, we present an approach to define the feature models with the Semantic Web technologies in order to ease the automation of the domain engineering and application engineering process.

As a future work, the results of this study will be used to support the utilization of a domain engineering methodology called SODSL which aims to provide an ontology-based approach to design and facilitate the domain-specific languages on service-oriented architectures [1].

7. References

[1] Bicer V., C. Togay,A. Dogru, "Service-Oriented Elearning Systems with Axiomatic Design" , *Integrated Design and Process Technology Conference*, San Diego, California, USA, 2006.

[2] Czarnecki, K., S. Helsen, and U. Eisenecker, "Formalizing Cardinality-based Feature Models and their Specialization", *Software Process Improvement and Practice*, John Wiley & Sons, 2005, v.10 pp.7-29.

[3] Czarnecki, K., S. Helsen, and U. Eisenecker, "Staged Configuration Using Feature Models", *Proceedings of SPLC 2004, Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, 2004, v.3154 pp.266-283.

[4] Czarnecki, K., and U.W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[5] Czarnecki, K., T. Bednasch, P. Unger and U. Eisenecker, "Generative Programming for Embedded Software: An Industrial Experience Report", *Proceedings of ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering 2002, Lecture*

Notes in Computer Science, Springer-Verlag Berlin Heidelberg, 2002, v.2487 pp.156-172.
[6] Greenfield, J., and K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools*, Wiley Boston, MA, USA, 2004.
[7] Horrocks, I., P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", *W3C Member Submission*, http://www.w3c.org/Submission/SWRL, 2004.
[8] Kang, K.C., J. Lee, and P. Donohoe, "Feature Oriented Product Line Software Engineering: Principles and Guidelines", *Domain Oriented Systems Development: Perspectives and Practices*, Taylor & Francis, UK, 2003, Ch.2.
[9] Lassila O., R.R. Swick, "Resource Description Framework(RDF) Model and Syntax Specification", *W3C Recommendation*, http://www.w3.org/TR/REC-rdf-syntax, 1999.
[10] Lee K., K.C. Kang, W. Chae, and B.W. Choi, "Feature-based Approach to Object-oriented Engineering of Applications for Reuse", *Software Practice and Exprience*, John Wiley & Sons, 2000, v.30 pp.1025-1046.
[11] Patel-Schneider P. F., I. Horrocks, F.v. Harmelen, "OWL Web Ontology Language 1.0 Abstract Syntax", *W3C Working Draft*, http://www.w3.org/TR/owl-absyn/ , 2002.
[12] Riebisch, M., "Towards a More Precise Definition of Feature Models", *Workshop on Modeling Variability for Object-Oriented Product Lines*, 2003.
[13] Smith, M.K., Welty, C., McGuinness, D.L., "OWL Web Ontology Language Guide", *W3C Recommendation*, http://www.w3.org/TR/owl-guide/ , 2004.
[14] Sochos P., I. Philippow, and M. Riebisch, "Feature-Oriented Development of Software Product Lines: Mapping Feature Models to the Architecture", *Object-Oriented and Internet-Based Technologies, Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, 2004, v.3263 pp.138-152.
[15] Togay, C., "HLA Tabanli Bileşenler ile Otomatik Uygulama Geliştirme", *II. Ulusal Yazilim Muhendisligi Sempozyumu (UYMS)*, pg: 243-251, 22-24, 2005.
[16]Togay, C., Dogru, A.H, "Federasyonların HLA Tabanlı Benzetimlere Tümleştirilme Otomasyonu için bir Mekanizma", *1. Ulusal Savunma Uygulamaları Modelleme Simülasyon Konferansı*, 2005.
[17] Togay, C., Dogru, A.H, "Infrastructure Design for HLA Based Automated Federation Development", *The Eighth World Conference on Integrated Design and Process Technology*, Beijing, China, pp: 698-704, 2005.
[18] Wang, H., Y.F. Li, J. Sun, H. Zhang, and J. Pan, "A Semantic Web Approach to Feature Modeling and Verification", *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, 2005.

Rol Modelleri ve Nesneye Yönelik Programlamaya Katkıları

Yunus Emre Selçuk, Nadia Erdoğan
İstanbul Teknik Üniversitesi, Elektrik Elektronik Fakültesi, Bilgisayar Mühendisliği Bölümü
selcukyu@itu.edu.tr, erdogan@cs.itu.edu.tr

Özet

Rol modelleri; dinamik sistemlerin nesneye yönelik programlama ile modellenmesinde karşılaşılan güçlüklerin çözümü için önerilen yollar arasında kullanışlı olmaları, NYP kavramları ile iyi uyuşmaları ve problemin çözümü için dolaysız bir yol sunmaları nedeniyle dikkati çekmektedir. Bu çalışmada Java diline rol desteği kazandıran bir rol modeli olan JAWIRO kullanılarak rollerin özellikleri tanıtılmış ve rol modellerinin yazılım kalite ölçütlerine sağladığı katkılar belirtilmiştir.

1. Giriş

Yaygın olarak kullanılan sınıf tabanlı nesneye yönelik programlama yaklaşımında, bir sınıf ile o sınıftan türetilmiş nesneler arasındaki ilişki kalıcı, değişmez ve dışlamalı bir yapıdadır. Bu durağan yapının NYP'ye kazandırdığı aynı tipten tüm nesnelere eş biçimli ulaşım avantajının yanı sıra, NYP kalıtım ve çok biçimlilik gibi önemli özelliklere sahiptir. Bütün bu güçlü yanlarına rağmen, NYP ile dinamik sistemlerin modellenmesinde çeşitli güçlükler ortaya çıkar. Modellenecek varlıkların ayrık sınıflara bölünebileceği ve bu varlıkların yeniden sınıflandırılmasına gerek duyulmayan durağan sistemlerde, NYP durağan yapısı sayesinde rahatça kullanılabilir. Ancak zamanla değişen ve yeniden sınıflanması gereken varlıklardan oluşan dinamik sistemleri modellerken, sistemdeki dinamik varlıkları NYP'nin durağan sınıf yapısını kullanarak modellemek zor olacaktır.

Dinamik olarak değişen ve evrimleşen varlıklar, zaman içerisinde çeşitli sorumluluklar üstlenir. Örneğin bir insan, hayatı boyunca birbirleri ile örtüşen çeşitli görevler alır: Öğrenciliğe başlayan bir kişi aynı zamanda yarı zamanlı bir iş sahibi olabilir; aynı anda eş, evlat, ebeveynlik gibi çeşitli yükümlülükleri yerine getirebilir. Dinamik sistemlerdeki varlıklar birbirinden farklı görevleri bağımsız olarak üstlenebilir.

Dinamik olarak evrim geçiren varlıkları modellemek için nesne düzeyinde özelleştirme yapılması, sınıf düzeyinde özelleştirmeden daha uygun olacaktır [1]. Bu durumda bir gerçek dünya varlığı, her biri yükümlü olduğu görevlerden birini yürüten bir rol nesnesi olmak üzere, birden fazla nesne ile temsil edilir. Bir nesnenin bir rolü, bu nesnenin diğer nesnelerin beklediği gibi davranabilmesi için gereken bir özellikler kümesi olarak tanımlanabilir [2]. Varlıklar modellenen sistemin kurallarına uygun bir şekilde roller kazanarak ve terk ederek türdeşlerinden farklılaşırlar. Bu şekilde evrim geçirmeye, nesne düzeyinde özelleşme adı verilmektedir. Rol modelleri ise, rollerin tasarımı için bir tarz ve rollerin kullanımı için çeşitli işlevler sunan yazılımlardır.

Sınıf düzeyinde kalıtım "aynısıdır" ilişkisini simgelerken, nesne düzeyinde kalıtım ise "bir parçasıdır" ilişkisini temsil eder [3]. Böylece, hem nesne düzeyinde özelleştirme sınıf düzeyinde özelleştirmeyi genişletir, hem de rol modelleri tüm NYP dillerinin felsefesine uyumlu özelliklere sahip olur. Bu nedenle bir çok başarılı rol modeli, mevcut bir NYP dilini genişletecek şekilde gerçeklenmiştir.

Rol modelleri sadece NYP'yi genişletmekle kalmaz, yazılım kalite ölçütlerine olumlu katkılarda da bulunur. Bu bildiride Java diline rol desteği kazandıran bir rol modeli olan JAWIRO (Java with Roles) adlı çalışmamıza değinilecek ve rol modellerinin NYP'ye katkıları incelenecektir. Bu konulardan önce rollerin özelliklerinin daha ayrıntılı olarak incelenmesi yerinde olacaktır.

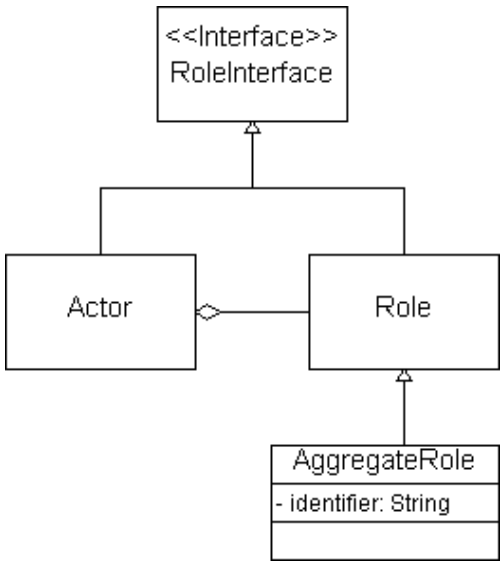
2. Rollerin özellikleri ve JAWIRO rol modeli

Rollerin özelliklerinin tanımlanması ve gruplanması, ortak temellere dayanmakla birlikte, değişik çalışmalarda çeşitli farklılıklar da içerir [2, 4]. Bu çalışmada, rollerin özellikleri temel ve ileri düzey olmak üzere iki gruba ayrılmıştır. Rollerin temel özellikleri, ağırlıklı olarak nesne düzeyinde özelleşme

ve bunu olası kılan özellikler ile ilgili iken; rollerin ileri düzey özellikleri ise büyük oranda JAWIRO'nun literatüre özgün katkılarından oluşan ve rollerden beklenebilecek diğer işlevleri sağlayan özelliklerdir. Bu çalışmada rollerin temel ve ileri düzey özellikleri ana hatlarıyla incelenerek JAWIRO ile kullanım örnekleri verilecektir. Daha ayrıntılı bilgi için Selçuk ve Erdoğan'ın bir çalışması [5] incelenebilir.

2.1. Rollerin temel özellikleri

Rol modelleri, nesne düzeyinde özelleşmeyi sağlamak üzere, bir gerçek dünya nesnesini sahip olduğu rollerin tümü ile tanımlar. Bununla birlikte, bir rol diğer bir rolü de oynayabilmelidir. Bu da rollerin çeşitli hiyerarşik düzenler oluşturacak şekilde ilişkilendirilebilmeleri anlamına gelir. Sonuç olarak, bir gerçek dünya varlığı, kökünde o varlığın zamanla değişmeyecek özelliklerini içeren bir nesne olmak üzere, birden fazla rol nesnesinin oluşturduğu bir *rol hiyerarşisi* ile modellenir. Bir A rolü B rolünü oynuyorsa A rolü *üst rol* veya *oyuncu*, B rolü de *alt rol* olarak adlandırılır. Şekil 1'de JAWIRO'nun rollerin kullanımı ile ilgili arayüzü görülebilir. JAWIRO'da rol nesneleri `Role` sınıfı ile, bir rol hiyerarşisinin kökü ise `Actor` sınıfı ile modellenir. Bu iki sınıfın benzer davranışları olabileceği için, `Actor` ve `Role` sınıfları `RoleInterface` adlı ortak bir arayüzü gerçekler.



Şekil 1. JAWIRO'nun kullanıcı arayüzü

Bir varlığın aynı rol tipinin birden fazla örneğine sahip olması gerekebilir. Böyle rollere bu çalışmada *nitelikli rol* adı verilmiştir. Şekil 1'de görüleceği gibi, `Role` sınıfından kalıtımla türetilen `AggregateRole` sınıfı nitelikli rolleri modeller. Nitelikli roller birbirlerinden bir niteleyici ile ayrılırlar.

Gerçek dünyada sınıf düzeyi özelleşme ile nesne düzeyinde özelleşme ilişkileri bir arada bulunabileceği gibi, JAWIRO rol modelinde de bu iki ilişki birlikte kullanılabilir. Bir rol sınıfından başka bir rol sınıfı türetilebilir ve hem üst sınıfın, hem de alt sınıfın örnekleri herhangi bir rol hiyerarşisinde yer alabilir.

Bir varlık farklı görevleri birbirinden bağımsız olarak yürütebileceği için, roller birbirlerinden bağımsız olarak kazanılabilmeli ve terk edilebilmelidir. Rol kazanımı için `RoleInterface.addRole`, rol terki için ise `Role.resign` metodu kullanılır. Bununla birlikte, bir varlık herhangi bir rolüne herhangi bir anda geçiş yapabilmelidir. *Rol geçişi* adı verilen bu eylem için `RoleInterface.as` metodu kullanılır. Rol geçişi ve rol hiyerarşisi sayesinde değişik roller bir çakışma olmaksızın aynı adlı üye metot ve değişkenlere sahip olabilir. Ancak rol geçişi yapmadan önce, bir varlık o rolü oynayıp oynamadığı hakkında sorgulanabilmelidir. Rol varlığı kontrolü adlı bu eylem için ise `RoleInterface.canSwitch` metodu kullanılır. Rol geçişi programcıya ilgili rol nesnesini gösteren bir işaretçi sağlar. Elde edilen işaretçi ile rollerin kullanımına sahibi üzerinden üye erişimi adı verilmiştir. Rollerin kullanımına ilişkin örnekler Şekil 2'de görülebilir.

```
//Nesne tanımları
ActorPerson ahmet, mehmet;
RoleDoctor docAhmet;
RolePatient patMehmet;
//Rol hiyerarşilerinin oluşturulması:
//1.Köklerin oluşturulması.
ahmet = new ActorPerson( "Ahmet Ata",
  "212-7778899");
mehmet = new ActorPerson( "Mehmet
  Mert", "212-7778899");
//2.Rollerin oluşturulması.
docAhmet = new RoleDoctor("212-
  2853300");
patMehmet = new RolePatient("599-
  5556677");
//3.Rollerin eklenmesi
ahmet.addRole(docAhmet);
mehmet.addRole(patMehmet);
//Rol varlığı kontrolü
if(ahmet.canSwitch("RoleDoctor") )
//Rol geçişi
((RoleDoctor)ahmet.as("RoleDoctor")).
  addPatient(patMehmet);
//Hasta iyileşip rolünü terk ediyor
patMehmet.resign();
```

Şekil 2. Rollerin temel özelliklerinin kullanımına örnekler

2.2. Rollerin ileri düzey özellikleri

Çeşitli dinamik sistemlerin gösterdikleri özellikler incelendiğinde, dinamik sistemlerin modellenmesinde rollerin temel özelliklerinin ötesinde gereksinimler de ortaya çıkacaktır. JAWIRO'da rollerin ileri düzey özellikleri bu tür gereksinimleri karşılamak üzere gerçeklenmiştir. Bu bölümde anlatılan özelliklerden kalıcılık ve aykırı rol ilişkilerinin önlenmesi dışındakiler, JAWIRO'nun literatüre özgün katkılarını oluşturmaktadır.

Kimi zaman bir sorumluluğun bir varlıktan diğerine aktarılması gerekebileceği için, JAWIRO'da bir rol alt rollerini kaybetmeden bir başka sahibe aktarılabilir. Bu özelliğe *rol aktarımı* adı verilmiştir ve `Role` sınıfının `transfer(RoleInterface newOwner)` adlı metodu bu amaçla kullanılır. Bazen de bir sorumluluk sonradan tekrar kullanmak üzere geçici olarak *askıya alınabilir*. Bu amaçla, ilgili rolleri gösteren bir işaretçi mevcut olduğunda kullanılmak üzere `Role.suspend` ve `Role.resume` metotları, role işaretçi bulunmadığı zaman kullanılmak üzere ise parametre olarak ilgili rolün sınıf adını alan `Actor.suspendRole` ve `Actor.resumeRole` metotları gerçeklenmiştir.

Rollerin bağımsız olarak kazanılıp terk edilmesi, askıya alınması ve başka sahibe aktarılması, sınırsız bağımsızlık anlamına gelmemelidir. Modellenen sistemin kurallarına bağlı olarak bu eylemlerin öncesinde veya sonrasında bazı işlemler yapılması, hatta bu eylemlerin onaylanmaması söz konusu olabilmelidir. Bu amaçla bir `Actor` örneğine `ConstraintStrategy` arayüzünü gerçekleyerek strateji tasarım kalıbına göre [6] ilişkilendirilen, *kısıtlama yöneticisi* olarak adlandırılan nesneler kullanılabilir. Bu şekilde *aykırı rol ilişkilerinin önlenmesi* mümkün kılınmıştır.

Bir rolü kullanmak, diğer bir deyişle o rolün bir üye metodunu çalıştırmak için tek yol rol geçişi değildir. Bir rol hiyerarşisinin herhangi bir üyesinin `RoleInterface.executeMethod( String methodName, Object... params)` metodu kullanılarak, adı ve parametreleri verilen metot çalıştırılabilir. *Ad ile üye erişimi* adı verilen bu özellik sayesinde roller tipten bağımsız olarak kullanılabilir. Örneğin Şekil 2'deki rol geçişi komutu yerine `ahmet.executeMethod("addPatient",patMehmet)` komutu da kullanılabilir. Eğer aynı hiyerarşideki katılımcıların birden fazlasının aynı imzaya sahip metodu bulunuyorsa, bu üyelerden en çok özelleşmiş, başka bir deyişle rol hiyerarşisinin en alt düzeyinde bulunan rolün üyesine erişim sağlanır. Ancak bazı rolleri *baskın rol* olarak işaretlemek üzere `RoleInterface.dominateSearch (boolean dominate)` metodu kullanılabilir. Bu durumda ad ile üye erişimi özelliği, en çok özelleşmiş rol yerine baskın rolün üyesine erişim sağlayacaktır.

Nesne düzeyinde çoklu kalıtım, rollerin JAWIRO'da gerçeklenen diğer bir ileri düzey özelliğidir. Bu özellik, bir rol örneğinin farklı sınıfların rol örnekleri tarafından oynanabilmesi anlamını taşır. Belli bir rol örneği aynı anda yalnız bir sahibe ait olabileceği için nesne düzeyinde çoklu kalıtım mantıksal bir belirsizlik ortaya çıkarmaz. Ad ile üye erişimi özelliği sayesinde de, alt rolün üst role tipten bağımsız olarak kolayca erişmesi sağlanmaktadır.

JAWIRO'da hem *danışma*, hem de *vekalet mekanizması* bir arada desteklenir. Rol geçişi sırasında mesajın ilk alıcısının saklanıp saklanmaması ile birbirinden ayrılan bu iki mekanizma gerçek dünya sistemlerinde bir arada görülebileceği için, bir rol modelinde de birlikte kullanılabilmelidirler. Rollerin JAWIRO ile gerçeklenen son ileri düzey özelliği ise *kalıcılık* yeteneğidir. Bu sayede bir veya daha fazla rol hiyerarşisi, tüm katılımcıları ile birlikte ve durum bilgilerini kaybetmeden sonradan geri yüklenmek üzere ikincil depolama ortamlarına aktarılabilir.

3. Rollerin yazılım kalite ölçütlerine katkıları

JAWIRO rol modelinin gerçeklenmesindeki amaç, dinamik sistemlerin etkin bir biçimde modellenebilmesi için Java dilinin rol desteği ile genişletilmesidir. Java dilinin seçilmesindeki neden, Java dilinin yansıtma gibi yeteneklerinin, rollerin bazı ileri düzey özelliklerinin üçüncü parti gereçler kullanmadan gerçeklenmesine olanak sağlamasıdır. Saptanan amacın sağlayacağı yarar ise; daha iyi modellemenin yazılım kalitesini iyileştirmesi, iyileşen kalitenin de yazılım projelerindeki yüksek başarısızlık oranını [7] azaltarak yazılım maliyetini düşürmesi olacaktır. Bu bölümde öncelikle rollerin temel ve ileri düzey özelliklerinin ortaya çıkardığı sonuçlar belirtilecek, ardından bu sonuçların yazılım kalite ölçütlerini nasıl etkiledikleri anlatılacaktır.

3.1. Rol kullanımının etkileri

Rollerin temel özelliklerinin kullanımı, tasarım sırasında *ilgi alanlarının ayrılması* ilkesine uyulmasını kolaylaştırır. İlgi alanlarının ayrılması ilkesi; yazılımın sadece belirli bir amaç, kavram veya hedef ile ilgili kısımlarının tanımlanabilmesi, birleştirilebilmesi ve değiştirilebilmesi yeteneğine işaret eder. Roller ise belirli görevleri yapmak için gerekli yeteneklerin

dinamik olarak kazanılmasına olanak tanıyarak, ilgi alanlarının ayrılması ilkesine uyulmasını kolaylaştırır [8].

Rollerin ileri düzey özelliklerinin, özellikle de ad ile üye erişimi ve baskın rollerin kullanılması ise yazılımın tiplere olan bağımlılığını azaltır. Bu iki özellik sayesinde, nesnelere ait oldukları tipten bağımsız olarak erişilebilir. Bir gerçek dünya varlığını simgeleyen rol hiyerarşisine bu şekilde gönderilen mesaj, o mesajı anlayabilecek rollerden en çok özelleşmiş olanı tarafından işlenir. Modellenen ortamın değişen şartlarına göre aynı mesajın sürekli en çok özelleşmiş rol tarafından değil de, bir başka rol tarafından cevaplanması gerekebilir. Bu durumda da gerçek dünya varlığının ilgili rolleri değişen koşullara göre baskın kılınır.

3.2. Rollerin etkilerinin kalite ölçütlerine katkıları

Rollerin temel özellikleri kullanılarak ilgi alanlarının ayrılması ilkesine uyulduğu takdirde; kod karmaşıklığı azaltılarak anlaşılabilirlik arttırılmış, ayrı amaçlara sahip bileşenlerin yeniden kullanılabilirliği arttırılmış ve bakım kolaylığı yükseltilmiş olacaktır [9]. Öte yandan, daha kaliteli bir yazılım üretmek için tek başına rollerin temel özelliklerinin kullanımı yeterli olmayacaktır. Bunun nedeni, rol geçişi işleminin yazılımın tiplere olan bağımlılığını arttırmasıdır. Rol geçişi yapabilmek için hem geçiş yapılacak rolün tipini bilmek, hem de rol geçişi komutunun geri döndürdüğü işaretçi üzerinde tip dönüşümü işlemi yapmak gereklidir (Şekil 2). Tiplere olan bağımlılığın artması yazılımın bakımını zorlaştıracağı için, strateji tasarım kalıbı [6] ile tiplere bağımlılık azaltılmalıdır. Bu açıdan bakıldığında, rol modelleri eksik bir araç olmakla eleştirilebilir. Ancak rollerin ileri düzey özelliklerinin kullanımı, söz konusu eksikliği giderecektir.

Rollerin ileri düzey özelliklerinin kullanımı ile tiplere olan bağımlılığın azaltılması, çokbiçimlilik derecesini de arttıracaktır. Çokbiçimlilik ise yazılım karmaşıklığının azaltılmasına yardımcı olarak okunabilirliği ve bakım kolaylığını iyileştirecektir [10]. Çokbiçimliliğin uygun kullanımı değinilen yararları sağlarken, çok yüksek veya çok düşük düzeylerde kullanımı ise yarardan çok zarar sağlayacaktır [11].

Rollerin ileri düzey özellikleri sayesinde tiplere olan bağımlılığın azalması, yeniden kullanılabilirliği de arttırır [12]. Yeniden kullanılabilirlik ise yazılım projelerinin maliyetini ve tamamlanma süresini azaltır [13]. Yeniden kullanılabilirliğin artması, yazılımın uyarlanabilirliğini ve bakım kolaylığını da arttıracaktır.

4. İlgili çalışmalar

Literatürde rol modelleri hakkında önemli bir miktarda çalışma mevcuttur. Bu çalışmalardan dinamik sistemlerin modellenmesine yönelik olanlar JAWIRO ile doğrudan karşılaştırılabilir. Bu bölümde incelenecek çalışmalardan ilki, Schrefl ve Thalhammer'a aittir ve [14] akademik kullanım amacıyla İnternet'ten indirilebilir. Gottlob, Schrefl ve Röck'ün Smalltalk'ta yaptıkları çalışmaya [1] dayanan ve Java ile yazılmış bu rol modelinde rollerin temel özelliklerinin tümü gerçeklenmiştir. JAWIRO'nun kullanıcı arayüzü ve rollerin temel özelliklerinin kullanımı Schrefl'in çalışmasına benzemekle birlikte, rollerin bu bildiride incelenen ileri düzey özellikleri Schrefl'in rol modelinde bulunmamaktadır.

Lee ve Bae'nin [15] geliştirilmiş rol modeli (GRM) olarak adlandırdıkları çalışmanın odak noktası, aykırı rol ilişkilerinin önlenmesidir. Bu amaçla bireysel rol nesneleri tek bir birleşik nesne oluştururlar. Bu yaklaşımın en önemli dezavantajı, bir rolün diğerini oynamasını olanaksız kılmasıdır.

Bettini ve arkadaşlarının çalışması olan DEC-JAVA [16], donatıcı tasarım dokusundan [6] esinlenerek hazırlanmıştır ve rollerin tanımlanması için kendi söz dizimine sahiptir. DEC-JAVA metotların dinamik olarak özelleştirilmesine dayanır ve henüz prototip sürecindedir.

C++ dilini rol desteği ile genişleten bir çalışma olan INADA [17], her tipin bir role karşılık geldiği kalıcı bir ortam sunar. Kalıcılık dışında rollerin ileri düzey özelliklerinin desteklenmediği bu çalışmada çalışma anında rol varlığı kontrolü bulunmamaktadır ve nitelikli roller desteklenmemektedir.

Rol modelleri üzerinde tasarım kalıpları düzeyinde bir çalışma [18] da Fowler tarafından yapılmıştır. Modellenecek sistemin gereksinimlerine göre önerilen çeşitli kalıplardan rollerin temel özelliklerini en kapsamlı şekilde içereni rol ilişkileri tasarım kalıbıdır. Rol ilişkileri kalıbı rol hiyerarşilerini desteklemez, bu kalıbı bir rolün diğer rolleri oynayabileceği şekilde genişletmek kod karmaşıklığını arttıracaktır. Bu artışın nedeni, başka rolleri oynayabilecek her rol tipi için bu kalıbın yinelenmesi gereğidir.

Şimdiye dek değinilen çalışmaların desteklenen özelliklere göre karşılaştırılması Tablo 1'de verilmiştir. Tablo 1'de görülebileceği gibi, rollerin temel ve ileri düzey özelliklerine verdiği geniş destekle JAWIRO, dinamik sistemlerin modellenmesi için önemli bir araç niteliği taşımaktadır.

Tablo 1. Güncel rol modellerinin rollerin gerçeklenen özellikleri açısından karşılaştırılması.

	JAWIRO	Schrefl, vd. [14]	Rol ilişkileri kalıbı [18]	DEC-JAVA [16]	Lee ve Bae [15]	INADA [17]
Gerçekleme dili	Java	Java	Java	Java	Java	C++
Nitelikli roller	+	+	+	+	–	–
Hiyerarşi desteği	+	+	–	+	–	–
Sınıf düzeyi ve nesne düzeyi kalıtımın birlikteliği	+	+	+	–	+	+
Çalışma anı rol varlığı kontrolü	+	+	+	–	–	+
Aykırı rol ilişkilerini önleme	+	–	–	–	+	–
Nesne düzeyinde çoklu kalıtım	+	–	–	–	–	–
Ad ile üye erişimi	+	–	–	–	–	–
Baskın roller	+	–	–	–	–	–
Danışma ve vekalet mekanizmalarını birlikteliği	+	–	–	–	–	–
Rol aktarımı	+	–	–	–	–	–
Rollerin askıya alınması ve askından indirilmesi	+	–	–	–	–	–
Kalıcılık	+	–	–	–	–	+

Rol desteğinin çalışma anı başarımına getirdiği ek yükün değerlendirilmesi ise Selçuk ve Erdoğan'ın bir çalışmasında [5] mevcuttur. Bu çalışmada JAWIRO ve Java ile başarım ölçümü programı yazılabilecek diğer iki çalışma olan Schrefl'in [14] rol modeli ile rol ilişkileri tasarım kalıbı [18] kullanılmıştır. Sonuç olarak, güncel bir donanım ve 10,000'den az rol nesnesi söz konusu olduğu sürece rol modellerinin getirdiği ek yükün 1 milisaniyenin altında olduğu gösterilmiştir.

5. Sonuçlar

Nesneye yönelik programlamanın tüm yeteneklerine rağmen, dinamik sistemlerin modellenmesi kolay bir problem değildir. Dinamik sistemlerin ve gerçek dünya sistemlerinin modellenmesinde karşılaşılan sorunları çözmek üzere çeşitli tasarım kalıpları oluşturulmuştur [6]. Bununla birlikte, kalite ölçütlerine göre 'iyi' yazılım üretmek ve literatürdeki tasarım kalıplarına hakim olmak, ancak zaman içerisinde kazanılan deneyim ile mümkün olabilir. Ancak JAWIRO kullanıcılara bir dinamik sistemi modelleyen 'iyi' bir yazılım üretmek için doğal ve kolay bir yol sunar.

Rol modelleri, dinamik sistemlerin etkin bir biçimde modellenmesinde kullanışlılıkları ile dikkat çeken bir araştırma alanı sunmaktadır [15]. Dinamik sistemleri modellerken nesne düzeyinde özelleştirme yapılması, sınıf düzeyinde özelleştirmeden daha uygundur [1] ve rol modellerinin öncelikli işlevi de nesne düzeyinde kalıtım yeteneği sunmaktır. Nesneye yönelik kalıtımın sınıf düzeyi kalıtımı doğal olarak genişletmesinden ve bu iki ilişkinin birbirini tamamlamasından dolayı [1], rol modelleri nesneye yönelik programlama yaklaşımına kolayca uyum sağlar.

Literatürde bulunan rol modellerinin yaygın olarak destekledikleri rol ilişkileri, dinamik sistemlerin en temel gereksinimlerinin çoğunu karşılamaktadır. Dolayısıyla sadece rollerin temel özelliklerine gereksinim duyan bir sistemin modellenmesi için JAWIRO tek seçenek değildir: Bu gibi sistemlerde herhangi bir rol modeli veya uygun tasarım kalıpları kullanılabilir. Bununla birlikte, literatürdeki rol modelleri roller ile gerçeklenebilecek her ilişkiyi desteklememektedir. Rollerin ileri düzey özelliklerine gereksinim duyan sistemlerde, sağladığı özgün katkılar nedeniyle JAWIRO tercih edilebilir. Aksi takdirde gereksinim duyulan özellikleri gerçekleme yükü programcının üzerinde düşecektir. Üstelik kuvvetli tiplemeye sahip dillerde, rollerin yalnızca temel özelliklerinin kullanımı tip bağımlılığını arttırarak bakım aşamasını zorlaştıracaktır. Bu zorlukların aşılması sorumluluğu da programcıya düşecektir. Oysa JAWIRO rollerin ileri düzey özelliklerini programcının kullanımına sunarak bu sorunu çözmekte ve yazılımın kalite ölçütlerine önemli katkılar sağlamaktadır.

JAWIRO halen geliştirmeye açık noktalar içermektedir. Bunlardan ilki kalıcılık yeteneğidir: İkincil depolama ortamında saklanan dosyalar 64-bit DES algoritması kullanılarak şifrelenmektedir. Strateji tasarım kalıbı [6] ile programcıya kendi şifreleyicilerini kullanma olanağı sağlanabilir. Bununla birlikte bir hiyerarşi ikincil depolama ortamına aktarıldığında, hiyerarşinin önceki hali silinmektedir. Bu durumun zorunlu değil seçimlik kılınması halinde, uygulamanın çeşitli aşamalarında, kalıcı ortamda yer alan önceki bilgiler kullanılarak bazı çözümlemeler yapılabilir. Kısıtlama yöneticisi JAWIRO'nun geliştirmeye açık ikinci noktasıdır: Kısıtlamalar ile

ilgili kurallar için hazırlanabilecek bir söz dizimi, ölümcül kilitlenmelere neden olabilecek kısıtlama kurallarının önceden belirlenmesi gibi yararlar sağlayabilir. Nitelikli roller de, niteleyicilerin tekilliğinin programcı tarafından değil de rol modeli tarafından denetlenmesi sağlanarak geliştirilebilir. Şimdiye dek söz edilen geliştirmeye açık noktalar JAWIRO'nun ana hedefi olan dinamik sistemlerin daha iyi modellenebilmesini engelleyecek eksiklikler değil, çoğu programcı tarafından da gerçeklenebilecek ikincil yeteneklerdir.

Roller genellikle belli bir bağlamda kullanılır. Bir varlık farklı bağlamlar ile ilgili rollere sahip olabileceği gibi, farklı varlıklar aynı bağlamda bulunan rollere sahip olabilir. Bağlamlar, aktörler ve roller arasında bulunan ilişkiler daha ayrıntılı incelenebilecek bir konu oluşturmaktadır. JAWIRO'ya bağlam desteğinin kazandırılması planlanan çalışmalar arasında bulunmaktadır.

JAWIRO rol modelini içeren Java paketine, kullanıcı arabiriminin dokümantasyonu ile birlikte, http://www.yunusemreselcuk.com/jawiro/index.html adresinden erişilebilir.

6. Kaynaklar

[1] Gottlob, G., Schrefl, M. ve Röck, B., "Extending Object-Oriented Systems with Roles", *ACM Trans. on Information Systems*, (14), 1996, pp. 268-296.

[2] Kristensen, B.B., "Conceptual Abstraction Theory and Practical Language Issues", *Theory and Practice of Object Systems*, (2), 1996, pp. 143-160.

[3] Zendler, A.M., "Foundation of the Taxonomic Object System", *Information and Software Technology*, (40), 1998, pp. 475-492.

[4] Schrefl, M. ve Thalhammer, T., "Using Roles in Java", *Software-Practice and Experience*, (34), 2004, pp. 449-464.

[5] Selçuk, Y.E. ve Erdoğan, N., "A Role Model for Description of Agent Behavior and Coordination", *Lecture Notes in Computer Science*, (3963), 2006, pp. 29-49.

[6] Gamma, E., vd., *Design Patterns Elements of Reusable Object Oriented Software*, Addison-Wesley, Boston, 1994.

[7] Peckham, J. (ed.), *Practicing Software Engineering in the 21st Century*, Idea Group Inc., PA, USA, 2003.

[8] Cabri, G., Ferrari, L. ve Leonardi, L., "Applying Security Policies Through Agent Roles: A JAAS Based Approach", *Science of Computer Programming*, (59), 2005, pp. 127-146.

[9] Ossher, H. ve Tarr, P., "Using Multidimensional Separation of Concerns to (Re)shape Evolving Software", *Communications of the ACM*, (44), 2001, pp. 43-50.

[10] NASA, *Software Quality Metrics for Object Oriented System Environments*, NASA Software Assurance Technology Center, 1996, pp. 22-23.

[11] Brito e Abreu, F. ve Melo, W., "Evaluating the Impact of Object-Oriented Design on Software Quality", *Proc. 3rd International Software Metrics Symposium*, Berlin, Germany, 1996, pp. 90-99.

[12] Barnard, J., *Discussion and Survey of Reusability Metrics, Report Reference: REUSABILITY–1 for the MOOD Project*, Department of Computer Science, University of Warwick, Coventry, UK, 1997.

[13] Terry, C. ve Dikel, D., "Reuse Library Standards Aid Users in Setting Up Organisational Reuse Programs", *Embedded Systems Programming Product News*, 1996.

[14] Schrefl, M. ve Thalhammer, T., "Using Roles in Java", *Software-Practice and Experience*, (34), 2004, pp. 449-464.

[15] Lee, J-S. ve Bae, D-H., "An Enhanced Role Model For Alleviating the Role-Binding Anomaly", *Software-Practice and Experience*, (32), 2002, pp. 1317-1344.

[16] Betini, L., Capecchi, S. ve Venneri, B., "Extending Java to Dynamic Object Behaviours", *Electronic Notes in Theoretical Computer Science*, (82), 2003.

[17] Aritsugi, M. ve Makinouchi, A., "Multiple-Type Objects in an Enhanced C++ Persistent Programming Language", *Software-Practice and Experience*, (30), 2000, pp. 151-174.

[18] Fowler, M, "Dealing with Roles", *yayınlanmamış makale*, http://martinfowler.com/apsupp/roles.pdf, 1997.

Specification and Analysis of Access Control for Outpatient Clinic Software Architecture

Selma Süloğlu
Middle East Technical University, Computer Engineering Department, Ankara, 06531, Turkey
e140797@ceng.metu.edu.tr
Cemil Ulu
Middle East Technical University, Computer Engineering Department, Ankara, 06531, Turkey
cemil@ceng.metu.edu.tr
Halit Oğuztüzün
Middle East Technical University, Computer Engineering Department, Ankara, 06531, Turkey
oguztuzn@ceng.metu.edu.tr

Abstract

We consider the access control, specifically the information confidentiality and integrity issues, for an existing outpatient clinic management system at the level of software architecture. First, we describe the software architecture of the information system with Wright, an architecture description language defined in R. Allen's dissertation. Second, we annotate the architecture description with confidentiality authorizations in Wright/c, the Wright ADL with extensions for lattice-based access control. We then check the configuration using the Wright/c analysis tool to detect and identify potential confidentiality problems.

1. Introduction

With transition to interconnected and decentralized configurations, as most information systems are built today, information security, which involves confidentiality, integrity and availability, has become a significant consideration in health management systems. In this paper, we focus on the confidentiality and integrity problems of an outpatient clinic management system, a subsystem of a commercially available hospital management system, from the software architecture point of view. Our working hypothesis is that formal software architecture descriptions provide a suitable medium to specify and analyze information security issues.

The theoretical background for this work, which spans lattice-based access control [1] and software architectures [2], is presented in C. Ulu's dissertation [3]. A follow up technical report is available from the Wright/c home page [4]. An exposition of architectural description with confidentiality in Wright/c appears in [5].

2. Case study

2.1. Scope of the study

This study is concerned with the outpatient clinic and medical services from the information security, in particular confidentially and integrity, perspective at the level of software architecture. The software architecture of the outpatient clinic management subsystem is modeled with Wright/c so as to specify and enable the static analysis of the confidentiality issues based on data flow over the architectural elements. To this end, the components and their interconnections are described, the hierarchical role structure is elucidated, and a clearance is assigned to each role. The access control lattice is constructed and the architecture configuration with confidentiality authorizations is expressed in Wright/c notation. Finally, the configuration is checked using the Wright/c analysis tool [3], for incongruous authorizations, which indicate potential violation of the Bell-LaPadula principles. The checking algorithm detects and reports excess privileges as well.

2.2. Outpatient clinic

The main function of an outpatient clinic management system is to support provision of health services, where patients, administrative staff including managers and secretaries, and medical staff including physicians, nurses, technicians and specialists are involved.

The components of the outpatient clinic management software form a tightly coupled system as can be observed from Figure 1. (An arrow from component A to component B indicates A's dependence on B.)

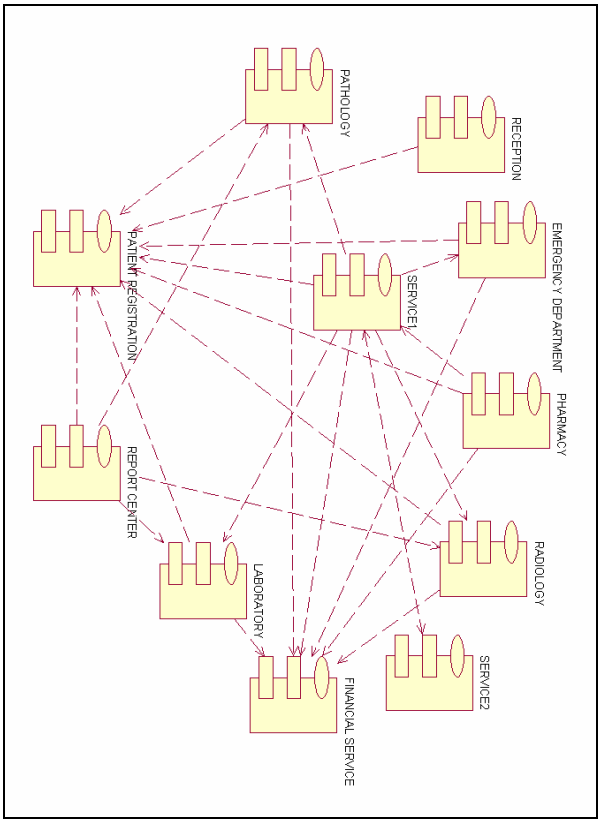


Figure 1. Component view of the outpatient clinic management software as a UML component diagram

Each *Service* component, such as *Service1*, offers health services such as tests, examinations, pre-diagnosis and diagnosis, consultation and treatment, and supports physicians to make consultation decisions by providing instant access to comprehensive patient information. *Service2*, which stands for any other service, has the same relations with other components as *Service1*, hence, only its relation with *Service1* is explicitly shown. Similarly to any *Service* component, *Emergency Department* serves emergency patients.

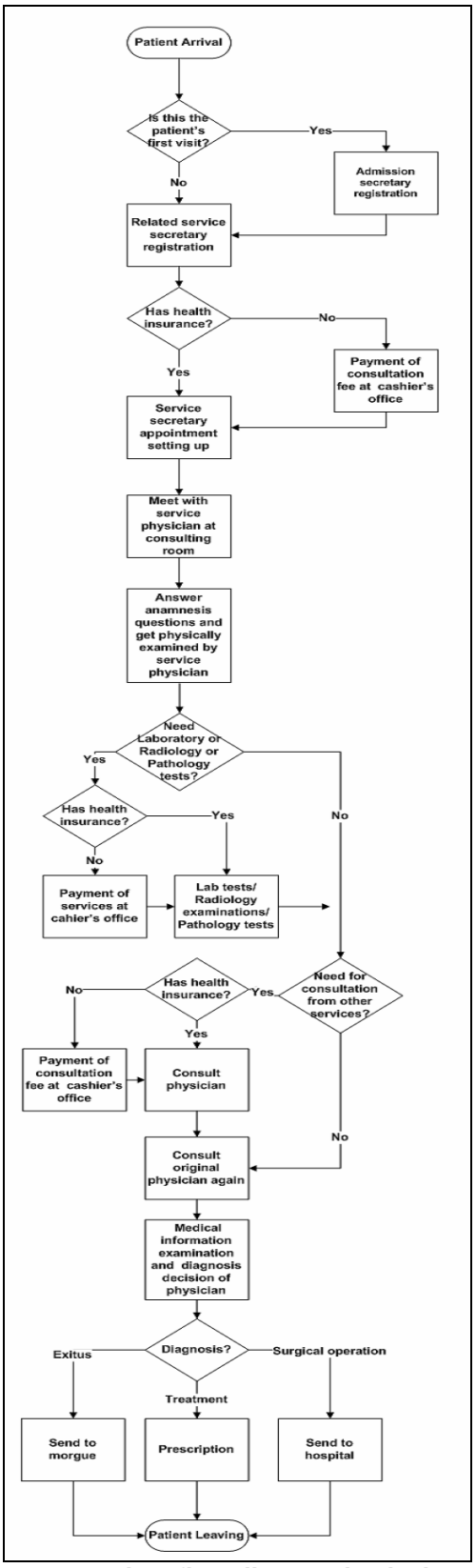


Figure 2. Patient flow diagram (typical case)

Pathology and *Laboratory* components handle specimen and test definitions, sample details, requested tests and test results. *Radiology* component deals with radiology test category definitions, requested examinations, examination results. *Pharmacy* component carries out pharmacy drug configuration, unit dosage facility, maintenance of drug inventory and supplier information. Moreover, it supports dispensing of medicine according to prescription content. *Reception* component records patient's short introductory, the physical location of service, administrative and medical staff and their names. *Patient Registration* component registers new patient's medical, personal, financial information, and handles complete patient information including his medical and judicial history and his sponsor details. *Report Center* component prepares various medical, personal and financial reports. *Financial Service* component handles definitions of health services provided, their prices, the insurance, agreement, sponsor and payment details, billing operations, and it calculates payment amount of the patient and the sponsor.

Figure 2 sketches the typical case of patient flow in the outpatient clinic.

Data items handled in the system are presented in Table 1.

Table 1. Data items

Data Group	*Abbrev.*	*Description*
Advisory	A1	Advisory (Patient's short introductory, physical location of service, medical and administrative staff, staff name and surname)
Financial	F1	Payment and Receivables, Invoice
	F2	Agreement and Sponsor Details
Medical	M1	Medical Reports
	M2	Pre-diagnosis and Diagnosis
	M3	Anamnesis Questions and Answers
	M4	Physical Examination Results
	M5	Prescription
	M6	Treatment
	M7	Patient Medical History-Secure Part
	M8	Patient Medical History-Public Part
Personal	P1	Patient Registry and Brevity (Name, Surname, Birth Date, Health Insurance, Sponsorship name, Address, Telephone, etc)
	P2	Patient Short Introductory (Name, Surname, Related Clinic Name)
	P3	Patient Judicial Record
Service	S1	Health Services Provided in Radiology(Examinations, drugs)
	S2	Health Services Provided in Laboratory(Tests)
	S3	Health Services Provided in Pathology (Tests)
	S4	Health Services Provided in Service (Treatment and consultation services, medical consumptions, such as lint, serum, etc.)
Test and Results	T1	Pathology test request
	T2	Pathology test results
	T3	Radiology examination request
	T4	Radiology examination results
	T5	Laboratory test request
	T6	Laboratory test results

Note that in determining the data items, their pairwise disjointness (in terms of contents) is taken into consideration from integrity point of view. From the point of view of confidentiality alone, we could handle inclusions of data items by assigning to the including item a higher (or equal) label than the included item.

2.3. Access control

The outpatient clinic information security, in particular confidentiality and integrity, problem is tackled with the help of role based access control model [6] and lattice based access control model based on mandatory access control (MAC) policy [1].

Data items presented in the preceding section take part in the data flow over the elements of the architecture, namely, the components and the connectors, as dictated by the behavioral descriptions thereof. The behavioral descriptions are formulated in CSP, a process algebraic notation due to C.A.R. Hoare, adopted in Wright as a sublanguage [7]. This enables the static analysis of the information flow over the architectural configuration subject to Bell-LaPadula principles for confidentiality analysis and Biba principles for integrity analysis. Indeed, data flow analysis is the heart of our Wright/c tool. For analysis the tool requires the description of the architectural configuration and the associated access control lattice prepared with the help of an XML editor. The tool consists of Wright/c parser (including CSP parser), analyzer and reporting modules. Wright/c parser and

CSP parser run in back and forth fashion to produce a standard ML source file for the Wright/c description of the configuration and the lattice. This source is combined with the analyzer source, which is also in ML. The analysis is then run on the ML interpreter. Figure 3 outlines the Wright/c tool structure and analysis process.

The analyzer mainly performs two tasks: a data flow analysis, and incongruity and excess privilege detection. The processes operate on the abstract form of the configuration with respect to the lattice model. Finally, the resulting analysis report is written to an output file. It includes incongruities with respect to the Bell-LaPaula principles, excess privileges or, if all goes well, an announcement of successful verification. An incongruity appears when any of the principles fail to be verified. If this is the case, the report includes warnings with the reasons (in terms of authorization level conflicts of the ports of component instances).

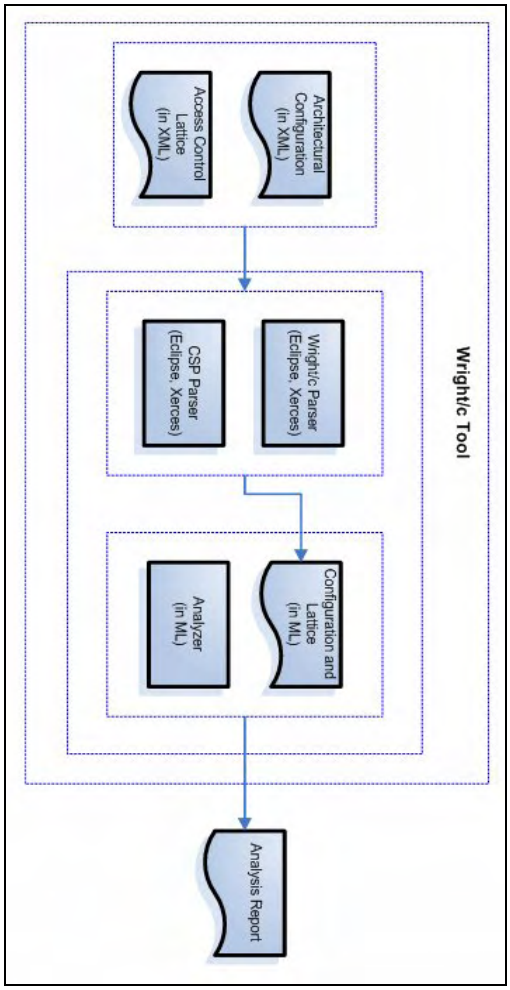


Figure 3. Wright/c tool structure and analysis process

The interested reader may consult [3] for a technical description including, in particular, the verification algorithm.

2.3.1. Role hierarchy and role rights. In the outpatient clinic management system there are thirteen defined roles, each of which represents responsibility of work with prescribed privileges. The roles' reading and writing privileges on data items, listed in Table 1, are presented in Table 2.

Table 2. Roles and their privileges

Role	*Has the right to read*	*Has the right to write*
Boss(B)	All items	No items
Administrator (AD)	All items	All items
Service Physician(SP)	A1,M*[1],P*,S4,T*	M1,M2,M3, M5,M6,M7, M8,S4
Service Nurse(SN)	A1,M4,M8,P1,P2	M4
Service Secretary(SS)	A1,P1,P2	T1,T3,T5,P3
Laboratory Technician(LT)	A1,P1,P2, S2,T5,T6	S2,T6
Radiology Technician(RT)	A1,P1,P2,S1, T3,T4	S1,T4
Pathology Specialist (PS)	A1,P1,P2, S3,T1,T2	S3,T2
Accountant(A)	A1,F1,F2,S*, P1,P2	F1
Pharmacist(P)	A1,M5,P1,P2	No items
Reception Secretary(RS)	A1,P2	No items
Admission Secretary(AS)	A1,F2,P1,P2	F2,P1,P2

A confidentiality clearance (designated with the prefix C) and an integrity clearance (designated with the prefix I) are assigned to each role as presented in Table 3.

Table 3. Confidentiality and integrity clearance assignment for the roles

Role	*Confidentiality Clearance*	*Integrity Clearance*
Boss (B)	CAD	IB
Administrator (AD)	CAD	IAD
Service	CSP	ISP

[1] M* stands for all data items M1 through M8, and so on.

Physician (SP)		
Service Nurse (SN)	CSN	ISN
Service Secretary (SS)	CSS	ISS
Accountant (A)	CA	IA
Laboratory Technician (LT)	CLT	ILT
Radiology Technician (RT)	CRT	IRT
Pathology Specialist (PS)	CPS	IPS

(Note that the roles of the Boss, the owner of the clinic, and the Administrator, the top manager, have been distinguished from the integrity perspective, although there is no need to distinguish them from the confidentiality perspective alone.)

2.4. Hierarchical role model and role rights

2.4.1. Confidentiality label assignment and access control lattice. Confidentiality labels (designated with the prefix CL) are assigned to data items according to their sensitivity, as presented in Table 4. Note that the existence of the top label CL1 is required for confidentiality labels to form a lattice.

Table 4. Assignment of confidentiality labels to data items

Data Item	*Confidentiality Label*
-	CL1
M1, M2, P3	CL2
M4	CL3
T1, T2	CL4
T3, T4	CL5
T5, T6	CL6
M5	CL7
M8	CL8
F1	CL9
S1	CL10
S2	CL11
S3	CL12
S4	CL13
F2	CL14
P1	CL15
P2, A1	CL16
M3, M7	CL17
M6	CL18

The access control lattice of the outpatient clinic system for confidentiality is presented in Figure 4. Nodes depicted as rectangles denote confidentiality labels. In the Bell-LaPadula(BLP) model of access control, a clearance that dominates a confidentiality label, say *CLx,* gives its holder the right to *read* all data items whose labels are "below" *CLx* (including *CLx* itself), and to *write* data items whose labels are "above" *CLx*. The former is known as the *simple security property*, and the latter as the **-property* [1].

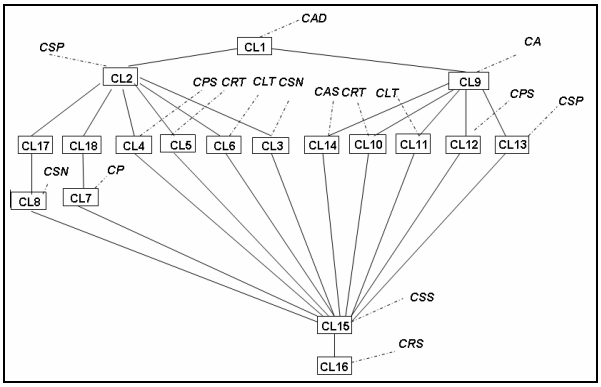


Figure 4. Access control lattice of the outpatient clinic for confidentiality

2.4.2. Integrity label assignment and access control lattice. Integrity labels (designated with the prefix IL) are assigned to data items according to their sensitivity, as presented in Table 5. Note that the existence of the bottom label IL12 is required for integrity labels to form a lattice.

Table 5. Assignment of integrity labels to data items

Data Item	*Integrity Label*
All data items	IL1
M1, M2, M3, M5, M6, M7, M8	IL2
S4	IL3
M4	IL4
T1,T3, T5	IL5
P3	IL6
F1	IL7
T6, S2	IL8
T4, S1	IL9
T2, S3	IL10
P1, P2, F2	IL11
-	IL12

The access control lattice of the outpatient clinic system for integrity is presented in Figure 5. Nodes depicted as rectangles denote integrity labels.

The Biba model is similar to BLP addressing integrity policy. A clearance that dominates an integrity label, say *ILx,* gives its holder the right to *write* all data

items whose labels are "below" *ILx* (including *ILx* itself), and to *read* data items whose labels are "above" *ILx*. The former is known as the *simple integrity property*, and the latter as the *integrity *-property* [1, 8]. (Note that to simplify the analysis we are adopting the dual of the conventional integrity lattice.)

The access control lattice structure of Outpatient Clinic for confidentiality and integrity, shown in Figure 4 and Figure 5, is readily expressible in Wright/c separately, only that for confidentiality given in Appendix.

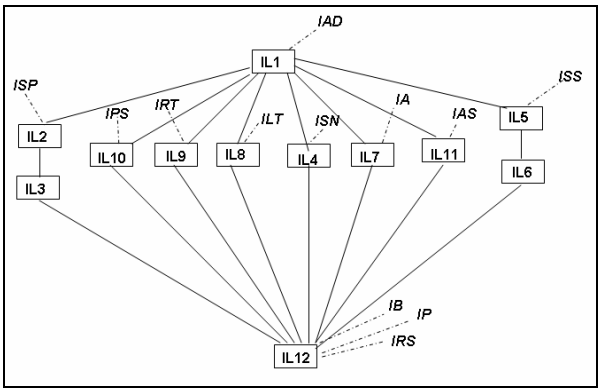


Figure 5. Access control lattice of the outpatient clinic for integrity

The Wright/c description of architectural configuration with confidentiality specifications is given, in outline form, in Appendix. The complete description is available as a part of a technical report accessible from the Wright/c Web page [4].

3. Analyzing the description

To illustrate the kind of reports that can result from the assignment of incongruous or excess privileges to ports, we include Figure 6. The reported case obtained by perturbing the assignments which we actually have. In fact, the report in Figure 7 reflects the actual situation.

The verification report in Figure 6 includes two types of warning. The "Potential confidentiality violation" warning point outs an authorization level conflict between ServiceA.CompleteIn and ServiceA.ResultsIn ports of component ServiceA. The "Excess privileges" warning indicates that the stated clearance cannot be fully exercised, so should be lowered if possible. The tool suggests a lower clearance.

```
VERIFICATION REPORT
**********
...

Component.Port: FinancialManagement.ServicesGivenIn type :INPUT_PORT clearance:CAD
  potentially output data security labels: NONE
  potentially input data security labels: CL11
                                          CL10
                                          CL12
                                          CL13
...

Component.Port: ServiceA.CompleteIn type :INPUT_PORT clearance:CSS
  potentially output data security labels: NONE
  potentially input data security labels: CL1

!!!! Security labels causing violation: CL1
...

Component.Port: ServiceA.ReportsIn type :INPUT_PORT clearance:CSS
  potentially output data security labels: NONE
  potentially input data security labels: CL2

!!!! Security labels causing violation: CL2
... .

Component.Port: Emergency.BrevityIn type :INPUT_PORT clearance:CP
  potentially output data security labels: NONE
  potentially input data security labels: CL15

...

WARNING :Potential confidentiality VIOLATION!..
 Please check the refused data security labels above...

 EXCESS PRIVILEGES
 **********

Excess privilege for FinancialManagement.ServicesGivenIn found:
  Current: CAD Recommended: CA
Excess privilege for Emergency.BrevityIn found:
  Current: CP Recommended: CSS

WARNING : Some excessive privileges are associated with ports as given above!..
 Please check them and revise your system configuration...
```

Figure 6. Verification report of confidentiality analysis with warnings

```
VERIFICATION REPORT
**********
...

Component.Port: FinancialManagement.ServicesGivenIn type :INPUT_PORT clearance:CA
  potentially output data security labels: NONE
  potentially input data security labels: CL11
                                          CL10
                                          CL12
                                          CL13
 ...

Component.Port: ServiceA.CompleteIn type :INPUT_PORT clearance:CAD
  potentially output data security labels: NONE
  potentially input data security labels: CL1
 ...

Component.Port: ServiceA.ReportsIn type :INPUT_PORT clearance:CSP
  potentially output data security labels: NONE
  potentially input data security labels: CL2

...
Component.Port: Emergency.BrevityIn type :INPUT_PORT clearance:CSS
  potentially output data security labels: NONE
  potentially input data security labels: CL15
...

********** The verification is SUCCESSFUL **********



 EXCESS PRIVILEGES
 **********
There is no excessive privileges...
```

Figure 7. Verification report of confidentiality analysis for the outpatient clinic software architecture

4. Specification and analysis in context

In the overall process, which addresses architectural design with security considerations, there are four phases: specification, presentation, analysis, incongruity-detection and correction. The last two phases can be applied iteratively until realizing a system architecture that is sound from confidentiality/integrity perspective.

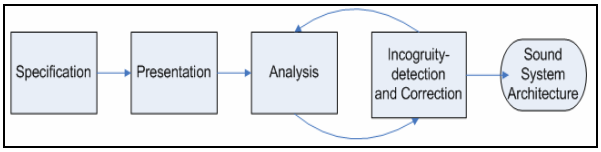


Figure 8. Phases of overall process

Specification phase includes determining components and connectors, forming the component view, building hierarchical role structure, identifying data items (if integrity analysis is done, the data items should be disjoint) that are exchanged between components of the system, matching clearances with roles and assigning security labels to data items.

Presentation phase involves constructing lattice using confidentiality/integrity clearances and confidentiality/integrity security labels and specifying system configuration in Wright/c notation [3].

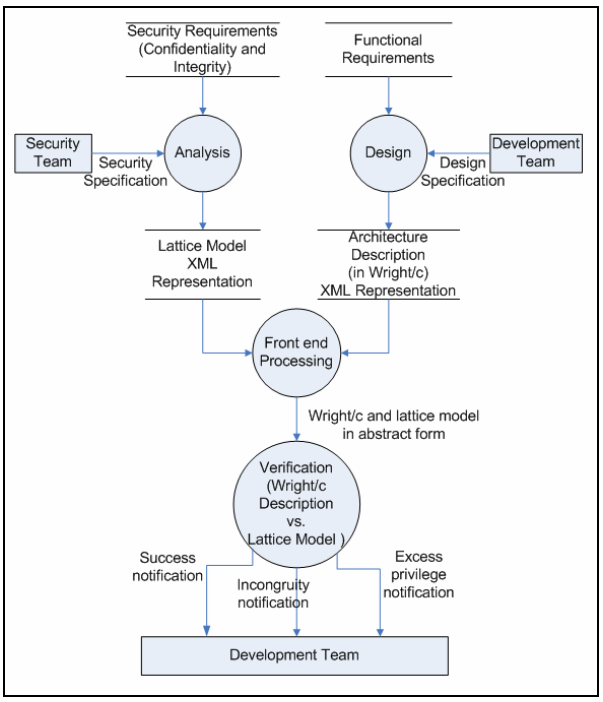


Figure 9. Data flow diagram for the specification and one iteration of verification process

Analysis phase comprises composing configuration with the help of Wright/c analysis tool, analyzing the configuration and assessing the analysis results presented as reports output by the tool.

In incongruity-detection and correction phase, detected excess privileges are reduced and reported flow incongruities are resolved and then analysis phase is applied again [3].

Data flow diagram for the specification and an iteration of verification process including development and security team roles and responsibilities within a software development organization is depicted in Figure 9 [3].

4. Discussion and conclusion

This work has successfully tested the formalism and the method put forth in [3] on a real-life example. A restriction on the present stage of the work is that architectural configuration and associated authorizations must be static.

The Wright/c description of the existing software architecture, confidentiality authorizations proved straightforward, although reverse engineering took six person-months of painstaking effort. Analysis on the current, un-optimized version of the tool suffered from run-time problems due to the size of the architectural configuration parameters, such as the total number of ports and number of security labels, although the analysis algorithm is polynomial. Supporting documentation of the present work is available from [4].

The confidentiality analysis revealed four cases of incongruity in the assignment of privileges, which were subsequently corrected, and one case of excess privilege (with no existing lower privilege to replace it).

Although Wright/c language and the associated tool do not specifically refer to integrity, its analysis is essentially same as that of confidentiality, and is not attempted in the present work.

6. Acknowledgements

The authors wish to express their gratitude to Şeref Arıkan of DataSel Bilgi Sistemleri A.Ş. for providing his expertise in the domain of hospital management systems. Thanks are also due to Ali Ferhat Tamur, Burak Yolaçan, Kurtcebe Eroğlu and Burcu Özbek for their contributions to the Wright/c analysis tool.

References

[1] Sandhu R.S., "Lattice-Based Access Control Models", IEEE Computer Vol. 26 No: 11, November 1993, 9-19.

[2] Allen Robert J., "A Formal Approach to Software Architecture", Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, May 1997.

[3] Ulu C., Specification and Verification of Confidentiality in Software Architectures", Ph.D. Thesis, School of Natural and Applied Sciences, Middle East Technical University, March 2004. Available as electronic resource from METU Library: http://library.metu.edu.tr/

[4] Wright/c Web Page: http://www.ceng.metu.edu.tr/~cemil/wrightc/

[5] Ulu C. and Oğuztüzün H. "Specification of Confidentiality Authorizations at Architecture Level", Proceedings of the 3rd Asia Pacific Symposium on Information Technology, January 2004, Istanbul, 482-489.

[6] Sandhu R.S., Coyne E.J., Feinstein H.L. and Youman C.E., "Role Base Access Control Models", IEEE Computer 29(2), February 1996.

[7] Hoare C.A.R, Communicating sequential processes, Prentice Hall International Series, 1985

[8] Biba K.J., " Integrity Considerations for Secure Computer Systems", Mitre TR-3153, Mitre Coorporation, Bedfotd, Massachusetts, 1977

Appendix: Wright/c confidentiality configuration and access control lattice(excerpts)

Lattice OC

Security Labels

CL1, CL2, CL3, CL4, CL5, CL6, CL7, CL8, CL9, CL10, CL11, CL12, CL13, CL14, CL15, CL16, CL17, CL18

Ordering

CL16, CL15, CL8, CL17, CL2, CL1
CL15, CL7, CL18, CL2
CL15, CL4, CL2
CL15, CL5, CL2
CL15, CL6, CL2
CL15, CL3, CL2
CL15, CL14, CL9, CL1
CL15, CL10, CL9
CL15, CL11, CL9
CL15, CL12, CL9
CL15, CL13, CL9
CL15, CL14, CL9

Clearance List

CAD	: CL1
CSP	: CL2
CA	: CL9
CPS	: CL4, CL12
CRT	: CL5, CL10
CLT	: CL6, CL11
CAS	: CL14
CSN	: CL3, CL8
CP	: CL7
CSS	: CL15
CRS	: CL16

End Lattice

Style Outpatient Management

Import Lattice OC "/project/OC/OC_lattice.txt"

Component Patient Registration ($\mu, \alpha, \chi, \gamma, \varphi$: Security Label)

…

Component PLR (χ, δ : Security Label)

…

Component Pharmacy ()

Port *BrevityIn* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *BrevityIn*

Port *PaymentIn* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *PaymentIn*

Port *PrescriptionIn* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *PrescriptionIn*

Computation= ($\overline{\text{BrevityIn.Re}\,ceive?x}$ || $\overline{\text{PaymentIn.Re}\,ceive?x}$ || $\overline{\text{PrescriptionIn.Re}\,ceive?x}$) $\rightarrow$ *SendMedicine* $\rightarrow$ *Computation*

Component Report Center (χ : Security Label)

…

Component Emergency Department (δ : Security Label)

…

Component Financial Management (χ : Security Label)

Port *BrevityIn* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *BrevityIn*

Port *AgreementEstablishmentIn*= $\overline{\text{Re}\,ceive?x} \rightarrow$ AgreeementEstablishmentIn

Port *ServicesGivenIn* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *ServicesGivenIn*

Port *PaymentOut* = $Send!x^{\chi} \rightarrow$ *PaymentOut*

Computation =

($\overline{\text{BrevityIn.Re}\,ceive?x}$ || $\overline{\text{ServicesGivenIn.Re}\,ceive?x}$ || $\overline{\text{AgreementEstablishmentIn Re}\,ceive?x}$) $\rightarrow$ *CalculatePayment*

$\rightarrow$ *PaymentOut.Send*$!x^{\chi} \rightarrow$ *Computation*

Component Service ($\alpha, \gamma, \varphi, \chi, \delta, \lambda$: Security Label)

…

Component Reception (χ : Security Label)

…

Connector *UniDirectionalConn*

…

Connector *BiDirectionalConn*

…

Connector *PoliDirectionalConn1*

…

Connector *PoliDirectionalConn2*

Role *SideA* = Send!x $\rightarrow$ *SideA*

Role *SideB* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *SideB*

Role *SideC* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *SideC*

Role *SideD* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *SideD*

Role *SideE* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *SideE*

Role *SideF* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *SideF*

Role *SideG* = $\overline{\text{Re}\,ceive?x} \rightarrow$ *SideG*

Glue = SideA.Send!x $\rightarrow$ $\overline{\text{SideB.Re}\,ceive?x} \rightarrow$ **Glue** □

SideA.Send!x $\rightarrow$ $\overline{\text{SideC.Re}\,ceive?x} \rightarrow$ **Glue** □

SideA.Send!x $\rightarrow$ $\overline{\text{SideD.Re}\,ceive?x} \rightarrow$ **Glue** □

SideA.Send!x → $\overline{\text{SideE.Receive?}x}$ → **Glue** □

SideA.Send!x → $\overline{\text{SideF.Receive?}x}$ → **Glue** □

SideA.Send!x → $\overline{\text{SideG.Receive?}x}$ → **Glue**

Connector *PoliDirectionalCon3*

…

End Style

Configuration Outpatient Clinic

Style Outpatient Management

Instances

…	…
FinancialManagement	FinancialManagement(OC.CL9)
Pharmacy	Pharmacy()
PCSecond1	PoliDirectionalConn2
…	…

Clearance

…	…
Pharmacy.BrevityIn	: CSS
Pharmacy.PaymentIn	: CA
Pharmacy.PrescriptionIn	: CP
FinancialManagement.BrevityIn	: CSS
FinancialManagement.ServicesGivenIn	: CA
FinancialManagement.AggreementEstablishmentIn	: CAS
FinancialManagement.PaymentOut	: CA
…	…

Attachments

…

FinancialManagement.PaymentOut **as** PCSecond1.SideA

Pathology.PaymentIn **as** PCSecond1.SideB

Laboratory.PaymentIn **as** PCSecond1.SideC

Radiology.PaymentIn **as** PCSecond1.SideD

Emergency.PaymentIn **as** PCSecond1.SideE

Pharmacy.PaymentIn **as** PCSecond1.SideF

ServiceA.PaymentIn **as** PCSecond1.SideG

…

Referans Mimarisi ve Uyarlanabilirlik

VERİ ERİŞİM VE YÖNETİM MİMARİSİ

Yeşim ATUN
ASELSAN A.Ş
yyeni@mst.aselsan.com.tr

Serap BOZBEY
ASELSAN A.Ş
bozbey@mst.aselsan.com.tr

ÖZ

Verilerin saklanması, bir grafiksel kullanıcı arayüzü ile sunulması ve bu arayüz kullanılarak veriler üzerinde işlem yapılması birçok uygulamanın ortak gerekleridir. Farklı uygulamalarda bu gereklerin karşılanması için aynı işler tekrar yapılmaktadır. Bu bildiride önerilen mimari ile veri tabanı işlemlerinin kolaylaştırılması, standartlaştırılması, veri tabanı yapısında olabilecek değişikliklerden uygulamaların en az düzeyde etkilenmesi ve uygulamaların veri tabanı yönetim sisteminden bağımsızlaştırılması amaçlanmıştır. Bu mimari her uygulamaya uyarlanabilen bir veri erişim ve yönetim alt yapısı hazırlanmasına olanak sağlamaktadır.

1. Giriş

ASELSAN Mikrodalga ve Sistem Teknolojileri (MST) grubunda REFORM bileşenleri kapsamında yürütülen çalışmada, uygulama destek servisi olarak veri tabanı yardımcı yazılımları ihtiyacı doğmuştur. Bu ihtiyacı gidermek için bir veri erişim ve yönetim mimarisi (VERY) gerçekleştirilmiştir. [5]

Bu bildiride, esnek ve modüler bir veri erişim ve yönetim mimarisi sunulmaktadır. Bu mimari, sunum ve veri tabanı erişim katmanından oluşmaktadır. Veri tabanı erişim katmanı, kendini kullanan uygulamaları veri tabanından soyutlayan bir ara katman yazılımıdır. Veri tabanı işlevlerini kendini kullanan uygulamalara sunar. Sunum katmanı ise, veriler üzerinde grafiksel kullanıcı arayüzleri üzerinden işlem yapılmasını sağlar. Grafiksel kullanıcı arayüzlerini daha genel hale getirmek, uygulamaların genel ihtiyaçlarını karşılamak amaçlı kullanılmaktadır. Eklenti mimarisini destekler. Uygulamalara özel istenen yetenekler eklenti mimarisi kullanılarak eklenmektedir.

Bu bildiride, ilk olarak mimariyi yönlendiren gereklerden bahsedilmektedir. Daha sonra VERY Mimarisi ve bu mimaride yer alan VERY Kütüphanesi ve VERY Kullanıcı Arayüzü (VERY KA) yazılımlarının genel mimarileri, kullanılan yöntemler ve tecrübeler anlatılmaktadır.

2. Veri Erişim ve Yönetim Mimarisini Yönlendiren Gerekler

Mimariyi yönlendiren gerekler aşağıda listelenmiştir:

2.1. Performans Gerekleri

Performans gerekleri mimarinin çıkarılmasında önemli bir etken olmaktadır. Zaman-kritik işlemlerde veri tabanı sorgu performansının iyi olması beklenmektedir. Performans kritik işlemlerde, veri tabanında sorgulama yapılırken sorgu sonuçları sayfa sayfa getirilebilmelidir.

2.2. Gerek Değişikliklerinde Esneklik

Yazılım mimarisi, gereklerin değişmesi durumunu göz önünde bulundurularak tasarlanmalıdır.

Veri Tabanı Kullanımında Esneklik

Uygulamaların değişik veri tabanları kullanma gereksinimi mevcuttur. Bu gereksinimi karşılamak için hazır bir alt yapı sistemine ihtiyaç vardır. Bu mimari sayesinde, değişik veri tabanlarını aynı arayüzler ile kullanmak mümkün olacaktır. Veri tabanında olabilecek değişiklikler (örneğin tablo isimlerinin değiştirilmesi, yeni kolonların eklenmesi, kolon isimlerinin değiştirilmesi gibi veri tabanı üzerinde sonradan yapılacak olan değişiklikler) uygulamaları etkilememelidir.

Bu alt yapıyı kullanan uygulamaların, veri tabanı yapılarını bilmeden veri tabanı işlemlerini

yapabilmeleri gereklidir. Bu sayede uygulama geliştiricilerin veri tabanı işlemlerini gerçekleştiren sınıfların bu işi nasıl yaptıklarını değil sadece ne yaptıklarını bilmeleri yeterli olacaktır.

Grafiksel Kullanıcı Arayüzlerinde Esneklik

Grafiksel kullanıcı arayüzlerinde gereklerin değişmesi kesinlikle kabul edilmesi gereken bir durumdur. Grafiksel kullanıcı arayüzü yazılımlarında en çok gerek değişikliği görünümle ilgili olmaktadır.

Görünümdeki değişiklerin yazılımdaki diğer kısımları etkilememesi beklenmektedir. Bu nedenle görünümle ilgili kısımlarla işlevselliğin ayrılması gerekmektedir.

2.3. Yeniden Kullanılabilirlik

Yeniden kullanılabilirliğin sağlanması için, farklı uygulamalarda aynı veri tabanı işlemlerinin tekrar tekrar yapılmasını engelleyen bir alt yapı olmalıdır.

2.4. Tekrarların Önlenmesi

Yazılımlar büyüdükçe eklenecek yeni bir özelliğin veya yapılacak bir değişikliğin maliyeti çok fazla olabilmektedir. Bu maliyetin nedeni aynı değişikliği birçok yerde yapmaktan kaynaklanmaktadır.

2.5. Aykırı Durumların Ele Alınması

Aykırı durumların ele alınması için, bölünmez işlem grupları (transaction) desteği olmalıdır. Yazılım mimarisi, veri tabanı işlemleri sırasında herhangi bir sorun olduğunda yapılan işlemleri geri alabilme yeteneğine sahip olmalıdır.

2.6. Kod izlenebilirlik ve kolay bakım

Paket oluşturma, kod izlenebilirliği ve kolay bakım yapma konusunda dikkat edilmesi gereken bir husustur.

Fonksiyonel olarak ilişkili paketler aynı paketlerde tutulmalıdır.

2.7. Test Edilebilirlik

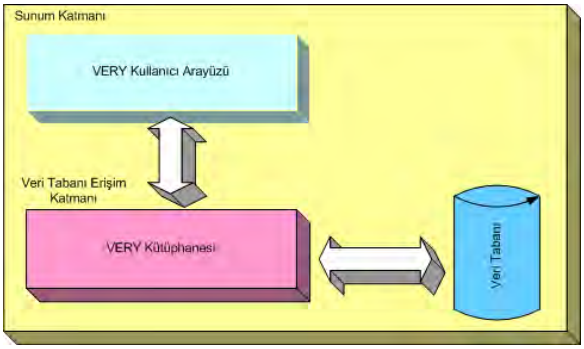
Mimari test edilebilir yapıda olmalıdır. Paketlerin test edilebilirliğini zorlaştıracak karmaşık bağımlılık ilişkileri olmamalıdır. Mimari kararlar test edilebilir olmalıdır. (Bkz. 4. Mimari Kararlar)

3. Veri Erişim ve Yönetim Mimarisi

Veri tabanı yazılımları, veri tabanı yönetim sisteminde (VTYS) tutulan verinin getirilmesi ve üzerinde işlem yapılması ihtiyaçlarını gerçekleştirirler. Bu ihtiyaçları karşılamak için kullanıcıya benzer arayüzler ve ortak çözümleri sunmak gerekir.

Bu amaçla, veri tabanı işlemlerinin kolaylaştırılması, standartlaştırılması, kodlamanın kullanılacak olan veri tabanı yönetim sisteminden bağımsız olarak yapılabilmesi ve veri tabanı yapısında olabilecek değişikliklerden en az düzeyde etkilenmesi için VERY mimarisi geliştirilmiştir.

VERY mimarisi genel olarak iki katmandan oluşmaktadır. (Şekil-1) Bunlardan VERY Kütüphanesi, veri tabanı yönetim sistemi üzerinde yer alan ve veri tabanından uygulamayı soyutlayan bir ara katmandır. Bu kütüphane, veri tabanı işlemlerinden ve veri tabanı ile nesneler arası eşlemeden sorumludur. İkinci katman ise VERY Kütüphanesi kullanarak verilere erişen bir kullanıcı arayüzü yazılımıdır. Veriler üzerinde işlemleri yönetir, sorgulama yapar, ekleme, güncelleme gibi işlemler için arayüz sunar. Bu sunum katmanın adı VERY KA'dır.



Şekil-1: VERY Genel Mimarisi

Veri Erişim ve Yönetim Mimarisinde her bir paket genişleyebilir, modüler, tekrar kullanılabilir olarak tasarlanmıştır. Bunu sağlamak amacıyla nesne-yönelimli tasarım kalıplarından yararlanılmıştır.[1][2]

VERY KA ile VERY Kütüphanesi arasında tek bir arayüzden kullanımı sağlamak amacıyla Abstract Factory tasarım kalıbı kullanılmıştır.[1][2]

3.1. VERY Kütüphanesi Yazılım Mimarisi

VERY Kütüphanesi, veri tabanı bağlantı kurma, genel veri tabanı işlemleri (ekleme, silme, güncelleme), sorgulama, sorgu saklama, saklanan sorgunun yüklenmesi, veri tabanı bilgilerinin taşınması, veri tabanı yapılandırma ayarları, veri tabanı yönetim işlemleri yeteneklerini gerçekleştirmektedir.

VERY Kütüphanesi, bir kütüphane olarak kullanılmasının yanı sıra bir sunucu yazılımı olarak da çalışmaktadır. VERY Kütüphanesini, sunucu olarak kullanmak isteyen yazılımlar soket alt yapısı üzerinden haberleşerek VERY Kütüphanesinin yeteneklerinden faydalanmaktadırlar. VERY Kütüphanesi, sunucu olarak, aynı anda birden fazla kullanıcıya hizmet verebilmektedir.

VERY Kütüphanesi Mimari tasarımının, esnek ve yeni yetenek eklenebilir olması sağlanmıştır. (Şekil-2) Öncellikle kütüphane olarak kullanılıyorsa, kendini kullanan uygulamaya bir arayüz sunmaktadır. Bu arayüz üzerinden kütüphanenin tüm yeteneklerine erişilmektedir.

VERY Kütüphanesi, veri tabanından ve işletim sisteminden bağımsız, veri tabanında yapılan değişikliklere uyum sağlayabilen bir yapıdadır. Bunun için Hibernate Nesne İlişkisel Eşleme (Object Relational Mapping) aracı kullanılmıştır. Veri tabanındaki tablolara ait XML eşleme dosyaları ve POJO (Plain Old Java Objects) sınıfları oluşturulmuştur. Veri tabanı bağlantı bilgileri yapılandırma dosyasında tanımlanarak ve Java'nın sunduğu JDBC sürücüleri kullanılarak veri tabanından bağımsızlık sağlanmıştır.

Hibernate alt yapısı, uygulamanın, herhangi bir VTYS ürünü ve veri tabanını kullanmasına olanak sağlamıştır. Veri tabanındaki tablolar, sınıflar olarak tanımlanıp, işlemler bu sınıfların nesneleri üzerinde yapılmıştır. Nesneler ve veri tabanı arasındaki ilişkiyi Hibernate alt yapısı düzenlemektedir. [3]

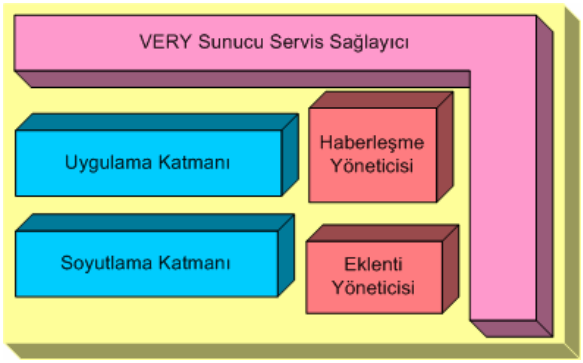
Farklı uygulamalar, VERY Kütüphanesini kullanacağı zaman, veri tabanı yapısını tanımladıktan sonra, veri tabanı bağlantı bilgilerini oluşturması ve tablolara karşılık gelen yapılandırma ve kaynak dosyaların üretilmesi yeterli olmaktadır. Veri tabanı yapısında bir değişiklik yapıldığında, tablolara ait XML dosyalarının ve POJO sınıflarının yeniden oluşturulması gerekmektedir. VERY Kütüphanesi ya da bu kütüphaneyi kullanan uygulamada bir değişiklik yapmaya gerek olmamaktadır.

VERY Kütüphanesi, uygulama ve soyutlama katmanlarından oluşmaktadır. Uygulama katmanında, VERY Kütüphanesinin yeteneklerini kendini kullanacak uygulamaya sunan bir dış programlama arayüzü bulunmaktadır. Bunun yanı sıra, her yetenek grubu için ayrı yöneticiler tanımlanmıştır. Bu yöneticilere erişim, VERY Sunucu servis sağlayıcı üzerinden yapılmaktadır. Her yetenek grubu için ayrı arayüzler (Interface) de tasarlanmıştır. Bu arayüzleri gerçekleyen sınıflardaki değişikliklerden, arayüzleri kullanan uygulamalar etkilenmemektedir. Böylece kod değişikliği ve idamesi kolay olmaktadır.

Soyutlama katmanında ise Hibernate alt yapısı ile birlikte XML dosyaları, POJO sınıfları ve veri tabanı bağlantı dosyaları yer almaktadır.

Uygulama katmanı, soyutlama katmanını bilmektedir. Ancak, soyutlama katmanı uygulama katmanını bilmemektedir. Uygulama katmanında yapılan değişiklikten soyutlama katmanı etkilenmemektedir. Bu da bağımlılığı azaltmaktadır.

VERY Kütüphanesine yeni özelliklerin eklenmesi için bir de eklenti mimarisi kullanılacaktır. Yeni yetenekler çekirdek VERY kütüphanesini değiştirmeyecek şekilde eklenebilecektir. Eklenti çıkarıldığında ise o yetenek çıkarılmış olacaktır. Böylece uygulamalara göre VERY Kütüphanesinin özelleştirilmesi sağlanacaktır. Bu da yeniden kullanılabilirlik için iyi bir örnektir.



Şekil-2 : VERY Kütüphanesi Yazılım Mimarisi

3.2. VERY Kullanıcı Arayüzü Yazılım Mimarisi

VERY KA, verilerin gösterilmesi, sorgu, verilerin taşınması, yedekleme/geri yükleme gibi işlemlerin kullanıcı ile etkileşimli olarak yapılmasını sağlamaktadır.

Verilerin gösterilmesi, sorgu yapılması, ekleme, silme, güncelleme işlemleri için VERY Kütüphanesi kullanılmaktadır.

Verilerin ekranda gösterilmesi için gerekli bilgilerin tutulduğu yapılandırma dosyaları kullanılmaktadır. Yapılandırma dosyalarında gösterilecek veri alanları ve

özellikleri yer almaktadır. Araç Çubuğu, sorgu penceresi ve liste görünümü gibi ekran öğeleri çalışma zamanında yapılandırma dosyalarından faydalanılarak oluşturulmaktadır.

VERY KA, sunum, uygulama ve soyutlama katmanlarından oluşmaktadır. Uygulama katmanında, VERY KA'nın yeteneklerini kendini kullanacak uygulamaya sunan VERY KA yöneticisi bulunmaktadır. Bunun yanı sıra, her yetenek grubu için ayrı yöneticiler tanımlanmıştır. Bu yöneticilere erişim, VERY KA yöneticisi üzerinden yapılmaktadır. Her yetenek grubu için ayrı arayüzler (Interface) tasarlanmıştır. Bu arayüzleri gerçekleyen sınıflardaki değişikliklerden, arayüzleri kullanan uygulamalar etkilenmemektedir. Böylece kod değişikliği ve idamesi kolay olmaktadır.

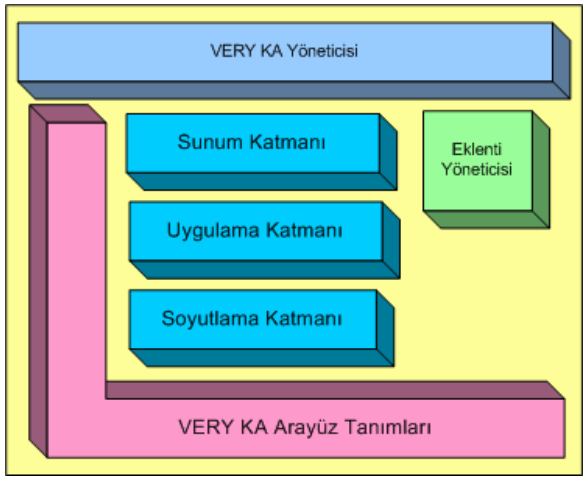
Sunum Katmanında, grafiksel kullanıcı arayüzü paketleri bulunmaktadır. Bu katmanda, Model-View-Controller (MVC) ve Model-View-Presenter (MVP) tasarım kalıpları kullanılmıştır. Bu kalıplar gerek değişikliklerinde esneklik, test edilebilirlik ve bakım yapılabilirlik sağlamak amacıyla kullanılmaktadır. [2]

Soyutlama katmanında ise VERY Kütüphanesi ile VERY KA tarafından kullanılan dosyaları yer almaktadır.

Sunum, uygulama ve soyutlama katmanları arasında yukarıdan aşağıya doğru bir bağımlılık söz konusudur. Aşağıdan yukarıya doğru bir bağımlılık yoktur. Uygulama katmanında yapılan değişiklikten soyutlama katmanı etkilenmemektedir. Soyutlama katmanında yapılan değişiklikten de sunum ve uygulama katmanları etkilenmemektedir.

VERY KA'ya yeni özelliklerin eklenmesi için eklenti mimarisi kullanılmaktadır. Yeni yetenekler, çekirdek VERY KA'yı değiştirmeyecek şekilde eklenebilmekte ve çıkarılabilmektedirler. [4]

VERY KA'yı kullanan uygulamalar kendilerine özel sorgu pencereleri tasarlamak istediklerinde, sorgu penceresi eklenti olarak tasarlanmakta ve eklenti yöneticisi tarafından işleme alınmaktadır. Araç çubuğuna yeni yetenek ekleme, detay penceresinin farklı görünümlerle tasarlanması ihtiyaçlarına göre eklentiler yazılmaktadır. Çekirdek VERY KA, eklenti yöneticisi sayesinde belirli bir dizinde bulunan eklentileri çalışma zamanında yükleyip kullanabilmektedir. [4]



Şekil-3: VERY KA Yazılım Mimarisi

4. Mimari Kararlar

Veri Erişim ve Yönetim mimarisinde uyulacak kurallar ve ilkeler aşağıdaki gibi olacaktır;

1. **Paketler arasında döngüsel bir bağımlılık olamaz.**

   Bu karar, gerek değişikliklerinde etkilenmemesi gereken paketler arasındaki ilişkiyi sağlamaktadır.

2. **Soyutlama katmanı sunum katmanına bağımlı olamaz.**

   Veri tabanı işlemlerinin yapıldığı soyutlama katmanı, sunum katmanı ile ilgili bir şey bilmemelidir. Sadece veri tabanı ile ilgili bilgi sahibi olmalıdır. Bu sayede sunum katmanındaki bir değişiklik soyutlama katmanını etkilemeyecektir. Grafiksel Kullanıcı arayüzünde yapılan değişikliklerden uygulamanın etkilenmemesi sağlanmış olur.

5. Mimari Kararların Testi

Bağımlılıkları kontrol etmek amacıyla JDepend aracı kullanılmıştır. JDepend, JUnit'le tümleşik çalışabildiği gibi ayrı bir uygulama olarak da çalıştırılabilir. JUnit'le entegre çalışabilmesinin avantajı, mimari kararların testinin yazıldıktan sonra otomatik olarak test edilebilecek olunmasıdır.

6. Mimarinin Gerçekleştirilmesi

VERY Kütüphanesi ve VERY KA, Java programlama dili kullanılarak yazılmıştır ve JAR formatı ile arşivlenmiştir.

Java dilinin seçilmesinin sebebi, işletim sisteminden bağımsız olması ve veri tabanından bağımsızlık için birçok çözüm sunmasıdır.

6.1. VERY Kütüphanesi Mimarisinin Gerçekleştirilmesi

Mimari gerçekleştirilmesine VERY Kütüphanesi ile başlanmıştır. Bu kütüphane, veri tabanı üzerinde çalışacak ince bir katman olarak gerçekleştirilmiştir. Bir servis sağlayıcı gibi hizmet verebileceği gibi, bir API kütüphanesi olarak da kullanılmaktadır.

VERY Kütüphanesi mimari tasarımı UML ile gerçekleştirilmiştir. Çevik programlama yöntemi ile çalışılmıştır.[6] Müşteriler tarafından belirlenen ihtiyaçlar, hikaye olarak haftalık iterasyonlarla gerçekleştirilmiştir. Gerçekleştirilen hikayelerin doğrulaması, müşteriler tarafından haftalık toplantılarda yapılmıştır. Fitnesse Wiki sayfası kullanılarak kabul testleri müşteriler tarafından hazırlanmıştır.[7] Testleri geçirmek üzere fixture kodları geliştirilmiştir.

VERY Kütüphanesinde, XML dosyalarının ve POJO sınıflarının uygulamanın ihtiyaçlarına uygun bir şekilde tanımlanması gerekmektedir. Tablolar arasında ilişkiler varsa, sorgu performansı yavaş olabilmektedir. Bunun için eşleme ve POJO dosyalarındaki tanımların ihtiyaca cevap verecek şekilde tanımlanması gerekmektedir.

XML dosyalarının ve POJO'ların oluşturulması için Hibernate alt yapısını kullanan MiddleGen aracı kullanılmıştır.

VERY Kütüphanesindeki çalışmalar çevik programlama yöntemiyle devam etmektedir.

6.2. VERY KA Kütüphanesi Mimarisinin Gerçekleştirilmesi

VERY KA, VERY Mimarisinde sunum katmanı olarak gerçekleştirilmiştir. (Şekil-1). Uygulamalar tarafından kütüphane olarak kullanılmaktadır.

Bu kütüphanede, verilerin gösterildiği ekran öğeleri konfigüre edilebilir şekilde hazırlanmıştır. Her liste penceresi için ayrı yapılandırma dosyası hazırlanmıştır. Veri alanları ve özellikleri ayrı ayrı tanımlanmıştır. Veri alanlarının tüm özellikleri, tipi, varsayılan değeri, basamak sayısı, hizalaması, genişlikleri, renkleri vs yapılandırma dosyasından ayarlanabilir duruma getirilmiştir.

VERY KA mimari tasarımı, VERY Kütüphanesinde olduğu gibi UML ile gerçekleştirilmiştir. Çevik programlama yöntemi ile çalışılmıştır.[6] Fitnesse Wiki sayfası kullanılarak kabul testleri müşteriler tarafından hazırlanmıştır.[7] Testleri geçirmek üzere fixture kodları geliştirilmiştir.

6.3. Genel Mimarinin Değerlendirilmesi

Hem VERY KA'nın hem de VERY Kütüphanesinin oldukça esnek olarak tasarlanması, dosyalarının sayısını artırmıştır. Birçok ihtiyaç, dosyalarından okunarak gerçekleştirilmektedir. Bu nedenle geliştiricilerin, ayarlarını doğru yapmaları gerekmektedir. Aksi takdirde yazılım hataları ile karşılaşmak muhtemeldir. Bu nedenle, dosyalarının ayarlarının ayrı bir grafiksel kullanıcı arayüzünden yapılabilmesi yeteneğinin de VERY KA kütüphanesine eklenmesi planlanmıştır.

7. Sonuç

Bu bildiride, ASELSAN MST grubunda gerçekleştirilen Veri Erişim ve Yönetim mimarisi anlatılmıştır. Veri Erişim ve Yönetim Mimarisinin gerçekleştirilmesi, her uygulamada veri tabanı işlemlerinin tekrar yapılmasının önlenmesini ve liste pencerelerinin standart grafiksel kullanıcı arayüzlerle sunulmasını sağlamıştır. Her uygulamada ayrı ayrı veri tabanı işlemlerinin yapılmasına ve liste pencelerinin tekrar kodlanmasına gerek kalmamıştır. Bu hem işgücü maliyetini azaltmış, hem de zaman kaybını engellemiştir.

Bu mimari sunum ve veri tabanı erişim katmanından oluşmaktadır. Veri tabanı erişim katmanı, veri tabanı işlemlerini soyutlayan VERY Kütüphanesidir. Bu katmanın üzerinde ise sunum katmanı yer almaktadır. Sunum katmanı veri tabanı üzerinde yapılacak grafiksel kullanıcı arayüzleri ve işlemlerini yöneten VERY KA'dır. Bildiride bu katmanlı mimari ve katmanların yapısından bahsedilmektedir. Mimarinin anlatılmasının ardından mimarinin gerçekleştirilmesi, kullanılan yöntemler ve tecrübeler anlatılmıştır.

8. Kaynak

[1] Robert C. Martin, www.objectmentor.com, *Design Principles and Design Pattern*

[2]] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sornmerlad, Michael Stal of Siemens AG, Germany., *Pattern Oriented Software Architecture Volume–1,*

[3] Cristian Bauer, Gavin King, *Hibernate In Action*

[4] Michael Pilone, *Plugins & Java, Dr. Dobb's Jounal*, December 2004

[5] Özgü Özköse Erdoğan, Baki Demirel, Alper Bostancı *Yeniden Kullanılabilir Kütüphane Geliştirme Yöntemi*

[6] Kent Beck, Martin Fowler, *Planning XP*

[7] www.fitnesse.org

UYUMLANABİLİR CBS YAZILIMI MİMARİ ÖNERİSİ

Ümit Demir
ASELSAN A.Ş.
udemir@mst.aselsan.com.tr

Koray Kadıoğlu
ASELSAN A.Ş.
kadioglu@mst.aselsan.com.tr

Öz

Yeni geliştirilen yazılım projelerinde, Coğrafi Bilgi Sistemlerinin (CBS) önemi ve kullanımı hızla artmaktadır. CBS, mekansal bilgileri kullanarak hızlı karar vermeyi sağlayan karar destek sistemleridir. Çeşitli alanlarda CBS yetenekleri barındıran projeler görmek mümkündür. Özellikle askeri projeler yaygın olarak CBS yetenekleri içermektedir. ASELSAN Mikrodalga ve Sistem Teknolojileri Grubu (MST) içinde geliştirilen birçok projede, harita uygulaması geliştirmek için çaba harcanmış ve benzer çalışmalar tekrar edilmiştir. Kendini tekrarlayan bu çalışmaları engellemek ve ortaklamak amacıyla, bir harita uygulama yazılımı mimarisi geliştirilmiştir. Bu mimarinin kullanıldığı Harita Altyapısı Sunum Platformu (HASP) yazılımının kullanlması ile farklı projelerin tüm genel ve özel ihtiyaçları karşılanabilmiş ve projelere özel harita yazılımları hızlıca hazırlanabilmiştir.

1. Giriş

ASELSAN MST grubunda REFoRM bileşenleri kapsamında yürütülen çalışmada, uygulama destek servisi olarak harita yazılımları ihtiyacı doğmuştur. [4] Bu ihtiyacı gidermek amacıyla HASP mimarisi hazırlanmıştır.

Bu bildiride, modüler ve genişleyebilir harita yazılımı mimarisi sunulmaktadır. Mimari, temel olarak sunum ve sunucu katmanlarından oluşmaktadır. Sunucu katmanı, CBS yeteneklerini içeren ve yeteneklerin detaylarını sunum katmanından soyutlayan katmandır. Harita işlevlerini kendisini kullanan uygulamalara sunar. Sunum katmanı ise harita işlemlerini kullanıcıya sunan, kullanıcı ile etkileşimi sağlayan katmandır. Harita yazılımı ekranlarının hızlı olarak geliştirilmesini sağlamaktadır. Sunum ve sunucu katmanı yazılımın genişlemesini sağlamak amacıyla eklenti mimarisi kullanmaktadır. Uygulamalara özel gerekler, eklenti mimarisi sayesinde karşılanabilmekte ve yazılıma eklenmek suretiyle yazılım genişletilmektedir.

2. HASP Mimarisini Yönlendiren Gerekler

Mimariyi yönlendiren gerekler aşağıda listelenmektedir:

2.1. Performans Gerekleri

Askeri uygulamalarda yüksek performans önemli bir gerektir. Gerçek zamanlı toplanan bilgilerin gerçek zamanlı sunulması gerekmektedir. Örneğin, çok sayıda harita ile çalışıldığı durumda gerçek zamanlı belirlenen hedeflerin harita üzerinde sunum performansının kabul edilebilir düzeyde olması gerekmektedir. HASP mimarisinde yüksek performans sağlanması hedeflenmiştir.

2.2. Gerek Değişikliklerinde Esneklik

HASP mimarisinde, gereklerin değişmesine mimarinin direnmemesi hedeflenmiştir. Mimarideki sunum katmanı, görsel değişikliklerin kolay ve hızlı yapılabilmesini hedeflemiştir. Yeni harita ekranlarının hazırlanması, farklı dillerin desteklenmesi, kullanılan menü ve tuşların özelleştirilmesi gibi görsel değişikliklerin hızlıca yapılabilmesi amaçlanmıştır. Bu değişikliklerin yazılım işlevselliğini etkilememesi önemlidir.

Sunucu katmanı, CBS yeteneklerinin kolayca değişmesini, yeni yeteneklerin eklenebilmesini hedeflemiştir. Bu katmandaki değişikliklerin sunum katmanını etkilememesi gerekmektedir.

2.3. Tekrar Kullanılabilirlik

Tüm harita uygulamalarında kullanılan bazı ortak işlevler bulunmaktadır. Örneğin, harita sunumu, harita üzerinde büyütme ve küçültme işlemleri hemen her

projede benzer ya da aynı şekilde gerçekleştirilen işlevlerdir. HASP Mimarisi, ortak işlevlerin yeniden kullanılabilirliğini sağlamayı amaçlamıştır.

2.4. Tekrarların Önlenmesi

Yazılımlar büyüdükçe eklenecek yeni bir özelliğin ya da yapılacak bir değişikliğin maliyeti çok fazla olabilmektedir. Bu maliyetin nedeni aynı değişikliği birçok yerde yapmaktan kaynaklanmaktadır. Örneğin kullanıcı arayüzlerinde, veri giriş alanları arka plan renginin beyaz olması istendiğinde, bütün pencerelerde bu değişiklik ayrı ayrı yapılmak zorunda kalınmamalıdır. HASP mimarisi, mümkün olduğunca kod tekrarlarını önlemeyi hedeflemiştir.

2.5. Kod İzlenebilirlik ve Kolay Bakım

Paket oluşturma, kod izlenebilirliği ve kolay bakım yapma konusunda dikkat edilmesi gereken bir husustur. HASP mimarisinde, işlevsel olarak ilişkili yeteneklerin aynı paketlerde tutulması amaçlanmıştır.

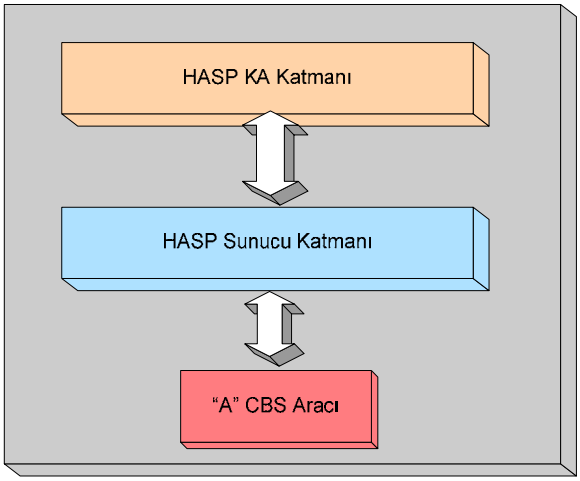
2.6. Test Edilebilirlik

Mimarinin test edilebilirliği önemli bir konudur. HASP mimarisi, mimarinin test edilebilir olmasını hedeflemiştir. Bu amaçla, paketlerin test edilebilirliğini zorlaştıracak karmaşık bağımlılık ilişkilerinin kurulmaması amaçlanmıştır.

3. Harita Altyapısı Sunum Platformu Mimarisi

Harita yazılımları, toplanan bilgilerin haritalar üzerinde gösterilmesi ve güncellenmesi işlemlerini gerçekleştiren yazılımlardır. Her projede yeniden harita uygulaması geliştirmek, yeni CBS araçları satın almak ve kullanmak zaman ve para kaybına sebep olmuştur. Bu amaçla, tüm harita uygulamalarının temel yetenekleri içeren, yeni yeteneklerin kolay ve takip edilebilir şekilde eklenmesine, hızlıca harita uygulamaları geliştirilmesine olanak sağlayan HASP Mimarisi geliştirilmiştir.

HASP Mimarisinde, katmanlı mimari tasarım kalıbı kullanılmıştır. [1] Mimari, sunucu katmanı ve grafiksel kullanıcı arayüzü (GKA) katmanı olmak üzere iki ana katmandan oluşmaktadır. Genel HASP mimarisi Şekil 1'de sunulmuştur. HASP Sunucu Katmanı, CBS yeteneklerini içeren ve bu yetenekleri harita yazılımlarından soyutlayan katmandır. Bu katman harita yeteneklerini gerçeklemekten sorumludur. Bu amaçla, hazır CBS araçları kullanımını desteklemektedir. HASP KA (sunum) Katmanı, sunucu katmanı yeteneklerini kullanarak, hızlıca harita uygulamaları geliştirmeyi sağlayan katmandır.



Şekil 1. Genel HASP Mimarisi

HASP Mimarisinde her bir paket genişleyebilir, modüler ve tekrar kullanılabilir olarak tasarlanmıştır. Bunu sağlamak amacıyla nesne-yönelimli tasarım kalıplarından yararlanılmıştır.[1][2]

3.1. HASP Sunucu Katmanı Yazılım Mimarisi

HASP Sunucu Katmanı, CBS işlemlerinin detaylarını kullanıcı arayüzü katmanından soyutlayan katmandır. Harita eklemek, harita üzerinde büyütme ve küçültme yapmak, mekansal sorgular yapmak, bilgileri haritalar üzerinde semboller kullanarak sunmak gibi harita işlemlerini gerçekleştirmek üzere arayüzler içermektedir. HASP KA Katmanı, sunucu katmanının bu arayüzleri üzerinden harita işlemlerini gerçekleştirmektedir.

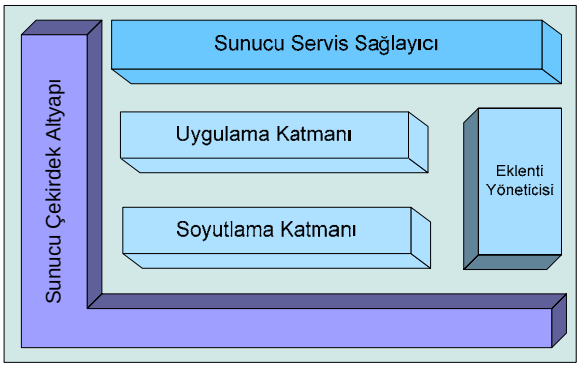
Bazı harita yetenekleri sunucu katmanı içinde gerçeklenmiştir. Bu sayede geliştirilen tüm harita uygulamalarında aynı algoritmaların kullanılması sağlanmıştır. Bazı temel yeteneklerin ise hazır CBS araçları ile karşılanmasının uygun olduğuna karar verilmiştir. Bu amaçla, CBS aracından beklenen yetenekler bir arayüz içinde toplanmıştır. Sunucu katmanı, CBS aracını, bu arayüzü gerçekleyen bir sürücü bileşeni üzerinden kullanmaktadır. Böylelikle CBS aracına bağımlılık zayıflatılmış ve kontrol edilebilir hale getirilmiştir. Şekil 2'de HASP Sunucu Katmanı yazılım mimarisi sunulmaktadır. Mimaride, uygulama katmanında, bir CBS aracı yöneticisi

bulunmaktadır. Bu yönetici hazır CBS araçlarının kullanımını sağlayacak olan yönetici bileşendir. Sunucu katmanı, çekirdek altyapısında tanımlanan arayüzler üzerinden CBS aracını kontrol etmektedir. Farklı CBS araçlarını kullanabilmek amacıyla, sunucu çekirdek altyapısında tanımlı arayüzleri gerçekleyen, hazır CBS aracının metot ve fonksiyonlarını kullanan bir sürücü bileşeni geliştirilecektir. Böylelikle yeni bir CBS aracına geçiş için, sadece yeni CBS aracına sürücü hazırlanması yeterli olacaktır. Bu da her projede farklı CBS araçlarını kolay bir şekilde kullanmayı mümkün kılmıştır.

HASP Sunucu Katmanı, harita uygulamalarında olabilecek tüm yetenekleri içermemektedir. Katman temel yetenekleri barındırmaktadır. Buna ek olarak yeni yeteneklerin hızlı ve kolay eklenebilmesine olanak sağlamaktadır. Genişleme amacıyla eklenti mimarisi kurulmuştur. Şekil 2'de eklenti yöneticisi gösterilmektedir. Bu yönetici, eklentilerin yazılıma dahil edilmesinden sorumludur. Örneğin çekirdek sunucu katmanında üç boyutlu analiz yetenekleri bulunmamaktadır. Yükseklik verisi okumak, görünürlük ve kesit analizi yapmak, arazi modellemesi üzerinde bilgilerin sunumunu gerçekleştirmek gibi gerekleri çekirdek sunucu karşılayamayacaktır. Fakat bu yeteneklerin eklenebilmesi için eklenti altyapısı sunmaktadır. Üç boyut yeteneklerini gerçekleyen bir eklenti geliştirilmiş ve çekirdek altyapıya eklenmiştir. Böylece proje gereği yoksa üç boyut eklentisi teslim edilmemiş, eğer ihtiyaç varsa teslim edilmiştir. HASP mimarisi, sunucu katmanının, her proje için geliştirilen eklentilerle zenginleşmesini sağlamıştır.

HASP Sunucu Katmanı, değişik yetenekleri sunan yönetici bileşenlerden oluşmaktadır. Yöneticiler, ilişkili yetenekleri içeren ve sunan yazılım bileşenleridir. Sunucu katmanı yetenekleri, bu yöneticiler üzerinden sunulmakta ve bu da işlemlerin, sunucu kontrolünde yapılmasını sağlamaktadır. Sunucu katmanın tek bir dış arayüzü bulunmaktadır. KA katmanı, sunucu servis sağlayıcı bileşeni üzerinden, sunucu katmanının ilgili yöneticilerine ulaşmakta ve bu yöneticiler üzerinden yetenekleri kullanmaktadır. Örneğin, yazılıma üç boyut analiz eklentisinin eklenmesi durumunda, KA katmanı, sunucu servis sağlayıcısı üzerinden üç boyut analiz yöneticisine ulaşacak ve analizleri bu yönetici bileşen üzerinden gerçekleştirecektir.

HASP Sunucu Katmanı, KA katmanı kullanılmadan da ayrı bir kütüphane olarak kullanılabilecektir. Harita yazılımı ekranları hazırlayan herhangi bir uygulama içinden, KA katmanını kullanmaksızın, istenilen sunucu yetenekleri kullanılarak, uygulama geliştirmek mümkün olacaktır. Bu amaçla, sunucu katmanının KA katmanına herhangi bir bağımlılığı yoktur.



Şekil 2. HASP Sunucu Katmanı Yazılım Mimarisi

3.2. HASP KA Katmanı Yazılım Mimarisi

HASP KA Katmanı, harita yazılımlarının hızlı ve kolay olarak hazırlanmasını hedeflemektedir. Bu katman, haritaların sunulmasını ve harita işlemlerinin kullanıcı ile etkileşimli olarak yapılmasını sağlamaktadır.

HASP KA Katmanı yazılım mimarisi Şekil 3'te verilmektedir. Katman, sunum, uygulama ve soyutlama alt katmanlarından oluşmaktadır. Sunum katmanında, grafiksel kullanıcı arayüzü işlemlerini gerçekleştiren paketler bulunmaktadır. Bu katmanda, yoğunlukla Model-View-Controller (MVC) tasarım kalıbı kullanılmıştır. Tasarım kalıpları kullanımı, tasarımda esneklik ve kolay güncellenebilirlik sağlamıştır. [2] Sunulacak ekranlar, menüler, vb görsel öğeler bu katmanda yer almaktadır.

Uygulama katmanında, HASP KA Katmanının yeteneklerini dış uygulamalara açan KA yönetici bileşeni bulunmaktadır. Bu yönetici, HASP KA Katmanının kullanımını yönetmektedir. Uygulama katmanında, ayrıca, gruplanmış görevleri yerine getiren farklı yöneticiler bulunmaktadır. Her yönetici, özelleşmiş işlemleri gerçekleştirmekte ve diğer yöneticilerle KA yöneticisi üzerinden haberleşmektedir.

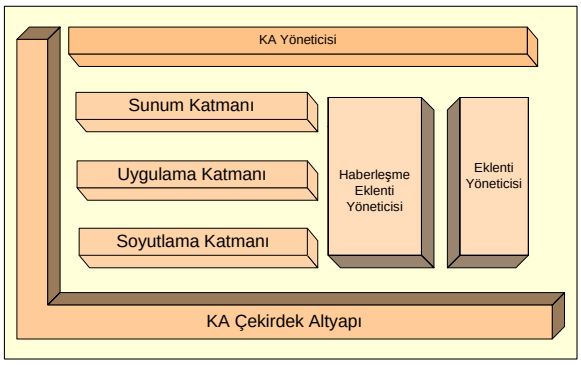
Soyutlama katmanında, HASP Sunucu Katmanını kullanan sunucu servis yönetici bileşeni bulunmaktadır. Bu yönetici, KA katmanının sunucu katmanına tek erişim noktasıdır. Böylece sunucu katmanına bağımlılık yönetilebilir hale getirilmiş ve sunucu değişikliğinin kolayca yapılabilmesi sağlanmıştır.

Sunum, uygulama ve soyutlama katmanları arasında yukarıdan aşağıya doğru bir bağımlılık söz konusudur. Aşağıdan yukarıya doğru bağımlılık yoktur. Uygulama katmanında yapılan bir değişiklikten soyutlama katmanı etkilenmemektedir. Soyutlama katmanında yapılan bir değişiklikten de sunum ve uygulama katmanları etkilenmemektedir.

HASP KA Katmanı, farklı projelerde kullanılabilmesi amacıyla, kolay uyumlanabilir bir yapıda geliştirilmiştir. Ayarlamalar, uyum dosyaları üzerinden yapılmaktadır. Özellikle, ana harita ekran boyutları, ana ekran açılış koordinatları, ekran renkleri gibi temel görsel ayarlar ve harita yazılımının ilk açılış ayarları gibi temel işlevsel ayarlar bu dosyalar üzerinden yapılabilmektedir. Bunlar, her harita yazılımı için olması gereken ayarlardır. Fakat tüm ayarları dosyalara yüklemek yazılımın tasarımını, kodlamasını ve uyarlanmasını zorlaştıracaktır. Çok sayıda dosya ile uğraşmak için, zaman içinde başka bir yazılıma ihtiyaç duyulacaktır. Bu noktada, yeni eklemeler ve güncellemelerin kolay yapılmasına olanak veren eklenti mimarisi kullanmak bu ihtiyacı karşılamıştır.[3] Çekirdek KA katmanında öngörülmeyen her yetenek eklenti olarak eklenmiştir. KA katmanı, her proje için eklentilerle zenginleştirilerek, yeni bir yazılım hazırlanmış ve projelere teslim edilmiştir. Bu mimari, sadece bir proje için geçerli olan bir gereğin, sadece o proje için karşılanabilmesine olanak sağlamıştır.

Genişleyebilir mimari sayesinde, bir projeye harita yazılımı hazırlanması gerektiğinde, HASP KA Katmanında bazı görsel ve işlevsel ayarlamalar yapılabilmiş ve gerektiğinde eklentiler geliştirerek, kısa sürede o projeye özel harita yazılımı hazırlanabilmiştir.

HASP KA Katmanı ayrı bir yazılım olarak geliştirilmiştir. Uygulama yazılımları ile tümleşmesi amacıyla haberleşme altyapısı içermektedir. Diğer uygulamalar ile mesaj alıp göndermek suretiyle haberleşmektedir. Haberleşmede kullanılabilecek değişik teknolojiler mevcuttur. Seçeneklerden bazıları soket üzerinden xml formatında mesajlar gönderip almak, CORBA haberleşmesi gerçekleştirmek, RMI kullanmaktır. Tek bir çözümü kullanmak, her proje için o teknolojinin desteklenmesi zorunluluğunu getirecektir. Fakat her projede farklı haberleşme teknolojileri tercih edilmiştir. Aynı zamanda, daha önceden geliştirilen projeleri de desteklemek gerekmiştir. Bu ihtiyaçlar, farklı haberleşme teknolojilerinin desteklenebilmesi ihtiyacını ortaya çıkarmıştır. Bunu sağlamak amacıyla haberleşme eklenti mimarisi kurulmuştur. Böylece her yeni teknolojiye geçiş için yapılması gerekenler belli bir noktada toparlanmış ve detaylandırılmıştır. Geliştirilen HASP yazılımında desteklenen teknolojiler, xml veri formatı kullanımı ve CORBA kullanımı olmuştur. Her teknoloji için bir adaptör hazırlanmış ve haberleşme altyapısına eklenmek suretiyle seçilen teknoloji kullanılabilmiştir.



Şekil 3. HASP KA Katmanı Yazılım Mimarisi

4. HASP Mimarisinin Gerçeklenmesi

Şekil 2 ve Şekil 3'te verilen mimari, UML kullanılarak tasarıma dökülmüştür. Tasarımı gerçeklemek üzere java programlama dili seçilmiştir. Özellikle, işletim sisteminden bağımsız olması, eklenti mimarisi gerçeklenmesinde sağladığı kolaylıklar ve zengin kod desteği sebebiyle java kullanımı iyi bir seçim olmuştur.

Geliştirme sürecinin başlangıcında, sunucu ve KA katmanlarının çekirdek altyapısını kurmak için önemli bir zaman harcanmıştır. Özellikle eklenti mimarisini kurmak amacıyla çalışma yapılmıştır. Bu zaman diliminde henüz proje gereklerine yönelik bir geliştirme çalışması yapılmamıştır. Buna karşın HASP katmanları için sağlam çekirdek altyapıları kurulmuştur.

Geliştirme sürecinde önemli olan bir konu, yazılım tasarımına bağlı kalmak olmuştur. Bu amaçla java geliştirme ortamları ile tümleşik olarak çalışabilen JDepend aracı kullanılmıştır. Tasarımdan kodlamaya geçildikçe oluşturulan paketler arası bağımlılık JDepend aracı ile kontrol altında tutulmuş ve mimariden sapılması engellenmiştir. Ayrıca, birim testlerinin oluşturulmasında kullanılan ve java ile tümleşik olarak çalışabilen JUnit aracı da yazılım birim testleri konusunda önemli destek sağlamıştır.

Proje gerekleri netleştikçe, bu gereklerin mimarideki yerleri belirlenmiştir. Bir yeteneği çekirdek kısma eklemek ya da eklenti olarak gerçeklemek kararı verilirken, yapılan alan analizi çalışmaları yön gösterici olmuştur. Bir gereğin tasarımda nerede

gerçekleneceği belirlenmiş ve HASP yazılımı buna uygun olarak geliştirilmiştir.

Kodlama süresince sunucu katmanı, KA katmanı ve eklentiler ayrı projeler halinde paralel olarak geliştirilmişlerdir. Ara zamanlarda tümleşim yapılarak, geliştirilen yazılımın sürekli çalışabilir olması sağlanmıştır. Geliştirme sürecinde kısmen çevik programlama uygulanmıştır. Proje gerekleri müşterilerden toplanarak, yazılımın projeler tarafından yönlendirilmesi sağlanmıştır. Hedeflendiği gibi her zaman çalışan ve projelerin ihtiyaçlarını karşılayan yeteneklere sahip bir yazılım bulundurulabilmiştir. Aynı anda birden fazla proje ile çalışmak ve farklı ihtiyaçları tek bir yazılımda gerçeklemek mümkün olabilmiştir. Herhangi bir zamanda, herhangi bir proje ile kolayca ve kısa sürede tümleşmek mümkün olmuştur.

5. HASP Mimarisinin Değerlendirilmesi

Mimariyi ilk gerçekleyen harita uygulamasının geliştirme süresi uzun sürmüştür. Bu süre çekirdek altyapı için harcanan zamanı da kapsamıştır. İlk uygulamadan sonra, daha kısa sürede harita yazılımı geliştirmek mümkün olmuştur.

Yazılımda kolay genişleyebilirlik sağlanmıştır. Her proje farklı yetenekler ile donatılan harita uygulamalarına sahip olmuştur.

Harita uygulama geliştirme süresi kısalmış ve paralel bir geliştirme mümkün olmuştur. Çekirdek yazılım ve eklentiler paralel olarak, farklı geliştiriciler tarafından geliştirilebilmiştir.

HASP yazılımına, yazılımı detaylı bilmeyen geliştiriciler tarafından da eklentiler yazılabilmesi ve değişiklikler yapılabilmesi sağlanmıştır.

Yazılım performansını arttırma konusunda başarılı sonuçlar elde edilmiştir. Eklenti mimarisi sayesinde, yazılıma sadece gerekli işlevlerin yüklenmesi sağlanmıştır. Aynı zamanda yazılımın çalışma zamanı performans kontrolü yapılabilmesi mümkün olmuştur. Çalışma zamanında kaynak yetersizliği sebebiyle gerçekleştirilemeyecek işlemleri içeren eklentiler açılıp kapatılarak yazılımın kararlı durumda kalması sağlanmıştır.

Hataların kontrollü olarak giderilebilmesi sağlanmıştır. Hatayı barındıran eklenti güncellenip çekirdek yazılıma eklenerek hatalar düzeltilebilir hale gelinmiştir.

Yazılım testleri parçalı olarak yapılabilmiştir. Çekirdek yazılım test edildikten sonra geriye test edilmemiş eklentiler kalmıştır. Her eklenti geliştirildikten sonra, sadece yeni eklenen yeteneklerin testi yapılmıştır. Eklentilerin çekirdek yazılımdaki güncelleme ve eklemeleri çekirdek yazılımın kontrolü altında yapıldığından, geriye hata kaynağı olarak eklentiler kalmıştır. Böylece aynı testlerin her projede tekrarlanması engellenmiştir.

HASP mimarisi sayesinde, farklı CBS araçlarının desteklenmesi, farklı harita formatlarının okunması mümkün hale gelmiştir.

Her proje teslimatında çekirdek yazılım ve sadece gerekli eklentiler teslim edilerek gereksiz kod ve yeteneklerin verilmesi engellenmiştir.

Çekirdek yazılımın, eklentileri çalıştırırken, önce eklentiler arası bağımlılığı çözümlemesi ve uygun olan eklentileri yüklemesi sayesinde üretilen yazılımın, mevcut yetenekleri ile her zaman kararlı olması sağlanmıştır. [3]

Bu kazanımların yanında geliştirme süresince bazı zorluklar da yaşanmıştır.

Özellikle, eklenti mimarisinin kullanılabilmesi için bazı arayüzlerin öğrenilmesi gerekmiştir.

Her eklenti için çok önemli olan, eklenti hakkında kritik bilgiler içeren eklenti tanım dosyalarının, her zaman güncel tutulması sorun yaratmıştır. Bu dosyalardaki bir yanlışlık, eklentilerin çalışmamasına sebep olacağından güncel olmaları kritik hale gelmiştir.

Eklentiler hakkında bilgi sahibi olunması gerekmiştir. Hangi gereklerin hangi eklentiler ile karşılanacağını bilmek, çalışma zamanında çıkabilecek sürprizleri engellemek açısından önem kazanmıştır. Eklentilerin yeteneklerini anlatan dokümanların hazırlanması gerekmiştir. Ayrıca altyapıya yeni eklentilerin nasıl yazılacağını detaylı olarak anlatan dokümanların da oluşturulması gerekmiştir.

Çekirdek yazılım ve her bir eklenti ayrı birer proje olarak geliştirildikleri için sürüm takibi kritik hale gelmiştir. Her yeni sürümün geçmişe yönelik uyumluluğu, diğer eklentiler ile uyumluluğu kontrol edilmesi gereken önemli bir konu olmuştur.

6. Sonuç

Bu bildiride, ASELSAN MST grubunda gerçekleştirilen Harita Uygulaması Sunum Platformu yazılım mimarisi anlatılmıştır. HASP Mimarisinin kullanılması, hızlı bir şekilde, her proje için projeye özel harita uygulamaları geliştirilebilmesini sağlamıştır. Her yeni harita yazılımında, sadece yeni yeteneklerin geliştirilmesi için çaba harcanmıştır. Bu hem işgücü maliyetini azaltmış, hem de zaman kaybını engellemiştir.

HASP mimarisi genel olarak sunum ve sunucu katmanlarından oluşmaktadır. HASP Sunucu katmanı, CBS yeteneklerini gerçekleyen katmandır. Sunum

katmanı ise sunucu katmanı yeteneklerini kullanarak kullanıcı ile etkileşimi sağlayan ve ekranlar sunan katmandır.

Bildiride katmanlı mimari ve HASP mimarisinde yer alan katmanların yapısından bahsedilmiştir. Mimarinin anlatılmasının ardından mimarinin gerçeklenmesi, kullanılan yöntemler ve mimarinin değerlendirilmesi anlatılmıştır.

7. Kaynak

[1] Robert C. Martin, *Design Principles and Design Pattern*, www.objectmentor.com

[2] [1] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley, 1996

[3] Ümit Demir, Koray Kadıoğlu, *Eklenti Mimarisi ve Yazılımlarda Eklenti Yönetimi*, 2006

[4] ASELSAN MST Grubu, *REFoRM – RADAR Elektronik Harp Fonksiyonel Referans Modeli*, 2004

ASELSAN K4İKG Projeleri İçin Teknik Referans Mimari ve Ortak Çalışma Ortamı

Bülent Durak[1],Ali Murat Topcu[2]
Tasarım Lideri, ASELSAN A.Ş. Mikrodalga ve Sistem Teknolojileri Grubu[1,2]
durak@aselsan.com.tr[1], topcu@aselsan.com.tr[2]

Özet

Komuta Kontrol Komünikasyon İstihbarat Keşif Gözetleme (K4İKG) faaliyetlerini otomatikleştirmek amacı ile ülkemizde uzun yıllardır askeri sistemler geliştirilmektedir. Bu sistemler tek bir araç üzerinde çalışan sistemden, sistemlerin sistemi (system of systems) olarak adlandırılabilecek birçok platformun dağıtık olarak bir arada çalışmasını gerektiren karmaşık sistemlere kadar değişkenlik gösterebilmektedir. K4İKG sistemleri yazılım yoğun sistemler oldukları için, sistemlerin özelliklerini ağırlıklı olarak bu sistemler üzerinde çalışan yazılımlar belirlemektedir. Bu geniş ürün ailesinde sistemlerin ihtiyaç duyduğu yazılımların, verimli ve hızlı bir şekilde geliştirilmesi, karşılıklı uyumlu çalışmaları, ölçeklenebilirliği ve taşınabilirliği gibi unsurların sağlanması için tüm sistemler tarafından kullanılan ortak bir çalışma ortamı (common operating environment) oluşturulması gerekmektedir. Bu bildiride ASELSAN'da geliştirilen K4İKG Sistemleri için açık sistemler ortamı oluşturulması konusunda yapılan araştırma ve uygulama çalışmaları anlatılmaktadır

1. Giriş

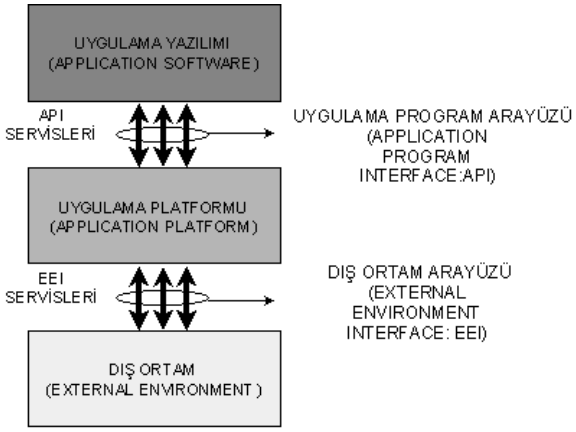
Değişik kurumlar veya bir kurum içerisinde değişik alt organizasyonlar tarafından geliştirilen yazılımların bir arada, karşılıklı uyumla çalışmasını sağlamak, taşınabilirliğini artırmak, geliştirme sürecindeki verim ve hızı artırmak için geliştirme çalışmalarının ortak prensiplere dayandırılması gerekmektedir. Teknik Referans Modeller (TRM), bilgi sistemlerinin teknik altyapılarının ortak prensiplere dayandırılarak geliştirilmesi için kullanılan araçlardan birisidir. TRM, bir sistemi oluşturan parçaların kavramsal gösterimi/tanımı için kullanılır. TRM, bir sistemi oluşturan olası tüm bileşenler, bileşenlerin sunduğu servisler ve aralarındaki ilişkileri tanımlar. Kurumlar, geliştirecekleri ürünlerin ortak prensiplere dayandırılması için TRM ile verilen tanımlamaları kullanarak Teknik Referans Mimarilerini oluşturur. Teknik Referans Mimaride bir gereksinim grubunu karşılamak üzere TRM'de tanımlanmış olan servislerin yazılım parçalarına eşlenmesi yapılır. Bir referans modelden birden fazla referans mimari oluşturulabilir.
Teknik Referans Mimari'de yer alan yazılım bileşenlerinden tüm projelerin ihtiyacı olan bileşenlerin gerçekleştirimi ile Ortak Çalışma Ortamı oluşturulur. Tüm projelerde Ortak Çalışma Ortamı kullanılmaya başlanması ile projelerin verimli ve hızlı bir şekilde geliştirilmesi, karşılıklı uyumlu çalışması, ölçeklenebilirliği ve taşınabilirliği sağlanmış olmaktadır.

İkinci bölümde literatürde teknik referans model ile ilgili kaynaklar sıralanacak, üçüncü bölümde yeni gelişmelerden bahsedilecektir. Sonuç bölümünde ASELSAN'ın bu konudaki yaklaşımlarına yer verilecektir.

2. Teknik Referans Modeller ve Teknik Mimariler

ASELSAN için Ortak Çalışma Ortamı oluşturulması için benzer sistemlerin geliştirildiği ABD ve NATO'daki çalışmalar incelenmiştir. ABD'de gerçekleştirilen Common Operating Environment ve NATO kapsamında geliştirilen NATO Common Operating Environment mimarileri, "IEEE Std P1003.0, Guide to POSIX Open Systems Environment" [1] standardı ile tanımlanmış Teknik Referans Model'e dayanmaktadır.

POSIX Açık Sistem Ortamı Modelinde, bir bilgi sistemi Şekil 1'de görüldüğü üzere 3 tip varlık (entity) ve bu varlıklar arasında iletişimi gerçekleştiren 2 tip arayüzden (interface) oluşmaktadır.



Şekil 1 : POSIX Açık Sistem Ortamı Modeli

Bu modelde bilgi sisteminin kendisi, uygulama yazılımı ile bu yazılımın üzerinde koştuğu uygulama platformundan oluşmakta, bu bölümler dışında kalan bütün unsurlar dış ortam olarak adlandırılmaktadır.

1. Uygulama Yazılımı: Çalışmak için Uygulama Platformu tarafından sunulan servislere ihtiyaç duyan her türlü uygulamaya yönelik yazılım bu sınıfta kabul edilmektedir. Her tür program, veri ve elektronik dokümantasyon bu bölümde yer almaktadır.
2. Uygulama Program Arayüzü (UPA): UPA, Uygulama Yazılımı ile Uygulama Platformu tarafından sunulan servisler arasındaki arayüz olarak tanımlanmaktadır. UPA 4 gruba bölünmektedir:
    - Sistem servisleri
    - Haberleşme servisleri
    - Bilgi servisleri
    - İnsan-makine etkileşim servisleri

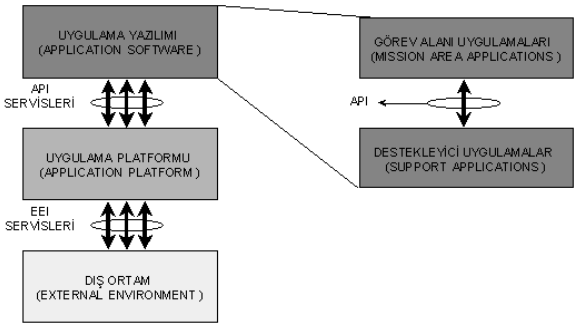
Uygulama Yazılımının, Uygulama Platformunun iç servislerine erişimi için Sistem Servisleri kullanılmaktadır. Diğer üç UPA grubu ise Uygulama Yazılımı'nın, ilgili Dış Ortam servislerine erişimi için kullanılmaktadır.

3. Uygulama Platformu (UP): Uygulama yazılımının çalışması için ihtiyaç duyduğu servisleri sunmaktadır. Uygulama Platformu, servislerini arayüzler aracılığı ile sunmaktadır. Bu sayede ortamın gerçekleştirimine yönelik ayrıntılar Uygulama Yazılımından soyutlanmaktadır.

4. Dış Ortam Arayüzü (DOA): Uygulama Platformu ile Dış Ortam arasında bilgi değişimi için tanımlanmış arayüzlerdir. DOA, 3 gruba bölünmektedir:
    - Haberleşme Servisleri
    - Bilgi Servisleri
    - İnsan-Makine Etkileşim Servisleri
5. Dış Ortam (DO): Uygulama Platformu'nun dışında kalan ancak Uygulama Platformu ile bilgi alıp veren her türlü birim bu sınıfa girmektedir.

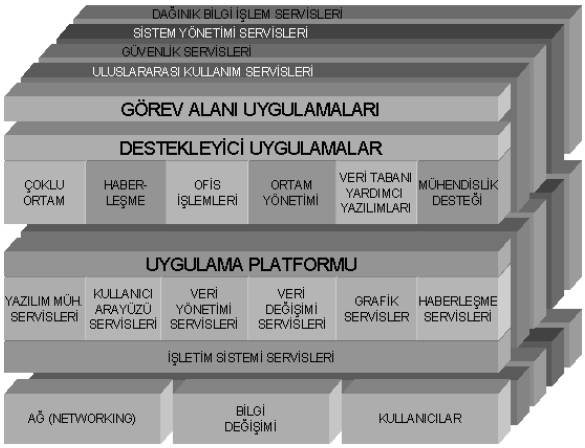
Bu standardın geliştirilerek askeri alana uyarlanması çalışmaları ABD Savunma Bakanlığı (DoD)-DISA tarafından başlatılmış ve 1992 yılında, DoD ve DoD ile bağlantılı olarak çalışan diğer stratejik savunma ve güvenlik teşkilatlarının bilgi yönetim sistemlerinde kullanılacak ortak bir teknik mimari tanımı getiren Technical Architecture Framework for Information Management (TAFIM) [2] dokümanının ilk sürümü yayınlanmıştır. Bunu 1994 ve 1996 yıllarında yayınlanan 2. ve 3. sürümler izlemiştir. TAFIM programının iptal edilmesinden sonra aynı kapsamdaki Teknik Referans Model çalışmaları DoD-TRM programı altında yürütülmeye başlamıştır ve günümüzde de bu çalışmalar sürmektedir.

TAFIM, POSIX-OSE referans modelini Şekil 2'de gösterilen biçimde geliştirmiş ve Uygulama Yazılımı kısmını kendi içinde 2 bölüme ayırmıştır.[3] Modelde, bu iki alt bölümün de kendi içlerinde alt bileşenlerine ayrılabileceği ve bileşenlerin kendi aralarında ve platformla UPA'lar üzerinden iletişim kurabileceği öngörülmektedir.



Şekil 2 : TAFIM Teknik Referans Modeli

Ayrıca TAFIM, POSIX-OSE'de sadece uygulama platformu için liste biçiminde verilen sistem servislerini genişleterek her bir servisi ait olduğu referans model bölümüne oturtacak biçimde Şekil 3'de gösterilen grafik yapıda sınıflandırmıştır.



Şekil 3 : TAFIM Teknik Referans Modeli Kapsamında Öngörülen Servisler

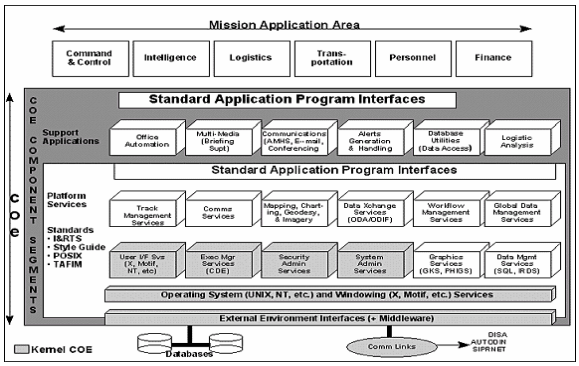
TAFIM modelinin getirdiği başlıca değişiklik "Destekleyici Uygulamalar"ı "Uygulama Yazılımı" düzeyinde tutmasıdır. Model, zaman içinde standartlaşan "Destekleyici Uygulamalar"ın, "Uygulama Platformu"nun bir parçası olarak aşağıya kaymasını öngörmektedir. Ek olarak, tüm model katmanlarında yansımaları olduğu kabul edilen "cross area" servislere (güvenlik, sistem yönetimi v.b.) Dağıtık Bilgi İşlem (Distributed Computing)" servisini eklemesidir.

DoD'de TRM ile verilmiş olan tanımlamaları kullanarak askeri sistemlerin geliştirilmesini hızlandırmak için temel yapı taşları (building block) adı verilen parçaların oluşturulması amacı ile çeşitli projeler başlatılmıştır.

ASELSAN'da ortak mimari oluşturma çalışmaları kapsamında yapılan incelemelerde değerlendirilen ilk mimari Common Operating Environment (DII COE) projesidir. DOD bu projenin K4İKG projelerinde kullanımı ile doğruluğu kanıtlanmış olan çözümlerin çeşitli projeleri arasında ortak olarak kullanılmasını ve maliyeti, proje risklerini düşürmeyi amaçlamaktadır [4]. DII COE, TRM'ye uygun olarak geliştirilmiş bir teknik mimari tanımı ile bu mimari tanıma uygun yazılım bileşenleri ve uygulama platformunun gerçekleştirimlerini içerir. Bu mimari kapsamında hazırlanan yazılım bileşenlerinin Uygulama Yazılımları tarafından nasıl kullanılacağına ilişkin standartlar, tanımlamalar ve kılavuzlar da sunar.

Mimari kapsamında geliştirilen tüm yazılım ürünlerinin donanımdan bağımsız olması ve POSIX uyumlu herhangi bir işletim sistemi üzerinde çalışması hedeflenmiştir.

POSIX, geliştirilmeye açık bir mimari tanımı içermektedir. Yeni bileşenler eklenmesi ve bu bileşenlerin DII COE'ye uygunluğunun değerlendirilmesi için yöntemler ve sorumluluklar da tanımlanmıştır.



Şekil 1 : Defense Information Infrastructure Common Operating Environment

İncelenen ikinci mimari, DoD ve DoD ile ilişkideki stratejik kurumlarca kullanılmak üzere ortaya konmuş TRM'yi referans alarak, gerek stratejik, gerekse taktik bilgi sistemlerinde kullanılacak Teknik Mimari olması planlanan Joint Technical Architecture projesidir(JTA) [5]. Bu proje kapsamında, bazı alanlara (domain) özel olarak kullanılacak endüstriyel ve askeri standartlar ile servis tanımları yapılmıştır. Tüm sistemlerde uygulanacak standartların belirlenmesinde endüstriyel ve askeri standartlar kullanılmıştır. JTA kapsamında herhangi bir gerçekleştirim veya ürüne referans verilmemektedir. Herhangi bir yazılım tanımı veya geliştirilmesi öngörülmemektedir. JTA'da ürün veya yazılım kullanımına ilişkin tek istisna, DII COE kullanımının K4İKG sistemlerinde zorunlu kılınmasıdır.

3. Servis Yönelimli Mimari ve Komuta Kontrol Projeleri

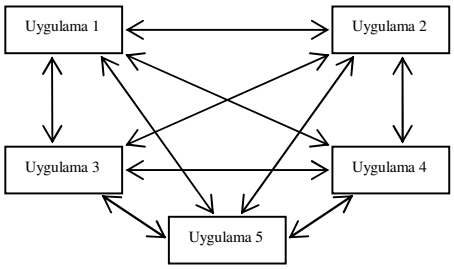
Özellikle Amerika ve NATO'daki çalışmalarda "Ağ Destekli Yetenek" kavramı öne çıkmıştır. Bu kavramda amaç veri ve bilginin isteyen ve yetkisi olan her birim tarafından nerede olursa olsun alınması ve kullanılmasıdır. Bu amaç için daha önce bahsettiğimiz mimarilerin tek başına kullanılması aşağıdaki nedenlerden dolayı yeterli değildir:

Karmaşıklık: Günümüz Komuta Kontrol sistemleri oldukça karmaşık yapılara ve çalışma ortamlarına sahiptir. Var olan sistemleri yenilemek çok pahalı olduğu için tekrar kullanılmaları gerekmektedir. Böyle bir ortamda birlikte çalışabilen sistemler üretmek oldukça zordur.

Tekrarlayan Uygulamalar: Her sistem kendi mimarisine uygun olarak aynı işlevleri gerçekleştiren

farklı fonksiyonlar içermektedir. Yeni bir sistem geliştirildiğinde var olan uygulamalar kullanılamazsa tekrarlayan uygulamalar problemine yeni bir ekleme yapılmış olur.

Arayüz Çeşitliliği: Birbirine doğrudan bağlanacak sistemlerin entegrasyonu n*(n-1) problem yaratmaktadır. Aşağıdaki örnekte her bir ok bir arayüzü belirtmektedir .Bu örnekte 20 bağlantı veya arayüz bulunmaktadır.Karşılıklı iki ayrı arayüz tek bir çizgi ile gösterilmiştir.



Şekil 4. Doğrudan n Uygulamanın Entegrasyonu

Bu entegrasyona bir sistemin daha eklenmesi, 2*(n-1) yeni arayüzün tanımlanması, gerçeklenmesi, test edilmesi, idame ettirilmesini gerektirecektir. Bu şekilde doğrudan entegrasyonun sistemlerin sistemi tipindeki projelerde arayüz çeşitliliğini artıracağı açıktır.[7]

Servis Yönelimli Mimari (SYM), güvenli bir ağ üzerinden birlikte çalışarak gerekli fonksiyonları sağlayan çeşitli yazılım servislerinden oluşan bir mimaridir. Bilgiyi sunan servisler ve bunları kullanan uygulamalar bu mimarinin temelini oluşturur..SYM'nin en önemli varsayımı servislerin çalıştıkları platforma bağımlı olmayabileceğidir. Bu servisler fonksiyonlarını işletim sistemi veya donanımdan bağımsız gerçekleştirebilmelidir. Bu sayede servis kullanıcıları servisin fiziksel yerini bilmek zorunda kalmaz. Komuta Kontrol projelerindeki dinamik birimler için bu özellik çok önemlidir. Servislerin ağ üzerinde yayınlanması ve standart protokoller kullanılarak gerçeklenmesi, tüm sistemin değişikliklerden etkilenmeden ve kesintiye uğramadan çalışması için önem arz etmektedir.

SYM ile ilgili NATO ve Amerika'da yürütülen çalışmalar takip edilmekte, ASELSAN'ın mevcut komuta kontrol yazılım mimarisinin bu yönde geliştirilmesi için çalışmalar devam etmektedir.

4. Sonuç

Yapılan incelemeler sonucu ASELSAN'da yapılan çalışmalarda ABD Savunma Bakanlığı (DoD) tarafından oluşturulmuş Teknik Referans Model'in kullanımı kararlaştırılmıştır.

ASELSAN'da K4İKG projeleri için ortak bir çalışma ortamı oluşturmak üzere yapılan çalışmalara 1997 yılında başlanmıştır [6]. Çalışmalara ASELSAN Common Operating Environment teriminin kısaltması olarak ASCORE adı verilmiştir. ASCORE çalışmaları 3 ana başlık altında incelenebilir:

Yazılım Bileşenlerinin Belirlenmesi: Önceki bölümlerde belirtildiği üzere DoD TRM referans alınmıştır. Seçilen Teknik Referans Modeline uygun olarak mevcut K4İKG projelerinin yanı sıra ileride alınması muhtemel projeler de göz önünde bulundurularak Uygulama Platformu ve Destekleyici Uygulamalar tarafından sunulması gereken servisler belirlenmiştir. Bileşen tabanlı geliştirme yöntemi kullanılacağı kabul edilerek servislerin yazılım bileşeni olarak gerçekleştirilmesi kararlaştırılmıştır.

Teknik Standartlar Profilinin Oluşturulması: ASCORE oluşturulurken karşılıklı uyumlu çalışmanın sağlanmasında en kritik nokta, bu servislerin üzerinde anlaşılmış standartlarca sağlanmasıdır. Bu amaca yönelik olarak ASELSAN tarafından K4İKG sistemlerinde kullanılması tavsiye edilen standartlar, Teknik Mimari Standartları Profili adı altında oluşturulmuştur. Teknik Mimari Standartları Profili tanımlanırken kullanılması öngörülen standartlar aşağıdaki başlıklar altında sınıflandırılmaktadır:

- Bilgi İşleme Standartları
- Bilgi Aktarım Standartları
- Bilgi Modelleme ve Bilgi Değişimi Standartları
- İnsan-Bilgisayar Arayüzü
- Bilgi Güvenliği

Yazılım Bileşenlerinin Gerçekleştirim Çalışmaları: Mevcut sistemlerdeki ihtiyaç durumu ve kritiklik derecesine göre belirlenen yazılım bileşenleri oluşturulmaya başlanmıştır. Yazılım bileşenlerinin Teknik Mimari Standartlar profiline uygun olarak geliştirilmesi gözetilmektedir.

Teknolojik gelişmeler yakından takip edilerek servis yönelimli yaklaşımların getirdiği gereksinimler de ASCORE mimarisine eklenmeye başlamıştır. Bu çalışmalar sonucunda karşılıklı birlikte çalışabilen, ortak bir platformu kullanan, ortak bir mimari ile geliştirilmiş ve taktik sahadaki tüm ihtiyaçları sağlayabilen sistem yazılımlarının daha uygun zamanda daha uygun maliyetle gerçeklenebilmesine olanak veren komuta kontrol yazılımları mimarisine ulaşılacaktır.

5. Kaynakça

[1] IEEE Std P1003.0, Guide to POSIX (Portable Operating System Interface) Open Systems Environment, IEEE, 1995
[2] Technical Architecture Framework for Information Management, DISA Center for Standards, 30 Haziran 1996,Version 3.0
[3] TAFIM Volume 2 Technical Reference Model, 30 Haziran 1996,Version 3.0
[4] Defense Information Infrastructure Common Operating Environment
[5] Joint Technical Architecture, Department of Defense, 3 Ekim 2003
[6] Z. Ayşın Zaim, Komuta Kontrol Yazılımlarında Mimari Modeller, Savunma Sanayi Sempozyumu, Ankara, 7-8 Kasım 2000
[7] Migrating to a service-oriented architecture Part 1, Part2, IBM Article 16 Dec 2003

Tasarım Desenleri

Nesne Tabanlı Yazılım Çerçevelerinde Tasarım Kalıpları

Rıza Horasan
Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği
horasan@itu.edu.tr

Özet

Benzer istekler için geliştirilen yazılım uygulamalarına temel oluşturan çerçeve kullanımı, uzun süredir kabul gören bir yöntemdir. Yazılım çerçevelerinin kullanımını destekleyen en önemli motivasyon, çerçevelerin tekrar kullanılabilirlik ve genişletilebilirlik özellikleridir. Çerçevelere bu özellikleri kazandırmanın temel yolu ise tekrar kullanılabilir ve genişletilebilir bir tasarımdır. Temel tasarım problemlerine çözüm önerileri sunan tasarım kalıplarının kullanımı, çerçevelere belirtilen özellikleri taşıyan tasarımların geliştirilmesi için uygun bir yöntemdir. Bu çalışmada üniversitelerin finans birimlerinin kullanması için geliştirilecek uygulamalara temel oluşturacak bir çerçevenin tasarım kalıpları kullanılarak geliştirilmiş bazı bileşenleri ve tasarım kalıpları kullanımının sağladığı avantajlar anlatılmaktadır.

1. Giriş

Nesne tabanlı yazılım çerçeveleri, nesne tabanlı uygulama geliştirmede genel bir teknoloji haline gelmiştir.[1] Çerçeve kullanımının temel yararı modüler, tekrar kullanılabilir, genişletilebilir, basit ve bakım yapılabilir uygulamalar geliştirmektir.[2] En sık kullanılan yazılım çerçevesi tanımına göre çerçeve, bir sistemin tamamının ya da bir kısmının tekrar kullanılabilir tasarımıdır.[3] Bu tanımdaki önemli nokta tekrar kullanımdır. Çünkü nesne tabanlı çerçeveler, uygulama geliştirme zamanını ve maliyetini önemli ölçüde azaltmaya yardımcı olan, önemli bir tekrar kullanım şeklidir.[4] Ancak bahsedilen özelliklere sahip bir çerçevenin tasarlanması zor bir süreçtir. Temel tasarım problemlerine çözümler sağlayarak tekrar kullanılabilir tasarımlar ortaya koyulmasını sağlayan tasarım kalıpları, çerçeve geliştirilirken karşılaşılan genel tasarım problemlerini çözmede yardımcı olabilir.[2]

Bu çalışmada yukarıda bahsedilen özelliklere sahip, üniversitelerin finans birimlerinde kullanılacak uygulamalar geliştirilirken, bu uygulamalara temel sağlayacak bir çerçevenin bazı bileşenleri anlatılmıştır. Çalışmanın amacı tekrar kullanılabilir ve genişletilebilir tasarıma sahip bir çerçeve geliştirilirken tasarım kalıpları kullanımının faydalarını incelemek ve tasarım kalıplarını uygulamada tecrübe kazanmaktır.

Çalışmanın 2. bölümünde nesne tabanlı yazılım çerçeveleri, 3. bölümünde ise tasarım kalıpları ile ilgili çalışmalar sunulmuştur. Geliştirilen çerçevenin çerçevenin bileşenlerinin anlatıldığı 4. bölümü, sonuçların ortaya koyulduğu 5. bölüm izlemektedir.

2. Nesne Tabanlı Yazılım Çerçeveleri

Yazılım mühendisleri, farklı müşteriler için geliştirdikleri ürünlerin, pek çok benzerliğe sahip olduğu ancak bu ortak noktalarının ortaya konmadığı hissine kapılırlar.[5] Özellikle, belirli bir problem uzayı(doğrulama ve yetkilendirme, bankacılık, insan kaynakları vb. uygulamaları) için geliştirilen uygulamalarda ortak pek çok nokta bulunmaktadır.

Üniversitelerin finans birimleri için geliştirilen uygulamaları düşündüğümüzde para, dekont, dağıtım, maaş gibi yapıların tüm uygulamalarda bulunduğu görülmektedir. Bunun yanında sadece bu bileşenlerin yapısal özelliklerinde değil, bileşenler arası ilişkilerde de benzerlikler bulunmaktadır. Ancak bahsedilen ortak yapıların detaylarında uygulamadan uygulamaya farklılıklar gösterdiği de ortadadır. Bu bağlamda, üniversiteler için geliştirilen finans uygulamalarının benzer özelliklere sahip olduğu söylenebilir.

Aynı problem uzayındaki uygulamaların ortak olan bileşenlerinin ve bileşenler arası ilişkilerinin her yeni uygulama için tekrar geliştirilmesi büyük bir zaman ve enerji kaybına yol açmaktadır. Bu kaybı azaltma çözümlerinden biri yazılım çerçeveleri kullanarak uygulama geliştirmektir.

Yazılım çerçeveleri temel olarak bir tekrar kullanma tekniğidir.[5] Çerçeve kullanımındaki temel nokta tasarımın tekrar kullanımıdır. [6] Bu tasarım, bir soyut sınıflar kümesi ve bu kümenin elemanlarının birbirleri ile olan ilişkileri üzerinden ortaya konur.[3] Yazılım çerçevelerinin soyut sınıflar kullanılarak

oluşturulmasının bir anlamı da genel tasarımı benzer olan uygulama bileşenlerinin, uygulama özelinde değiştirilebilmesidir. Bir çerçeve temel alınarak geliştirilen uygulama, çerçevenin küçük değişimler göstermesi ve üst seviyedeki tasarım özelliklerinin uygulama tarafından kabul edilmesi özelliklerini taşır. Diğer bir söyleyişle çerçevenin tekrar kullanılması hem tasarım hem de kod parçaları düzeyinde gerçekleşir.[5]

Bir çerçeve üzerine geliştirilmiş bir uygulamanın, uygulama özelinde değiştirilebilecek ve geliştirilebilecek yapılara sahip olabilmesi yazılım çerçevelerin genişletilebilir ve geliştirilebilir olması problemini ortaya çıkartmaktadır. Çünkü uygulama geliştiriciler özel bir uygulama için çerçeveyi, kalıtım özelliğini ya da çerçeve sınıflarından türemiş nesneleri birleştirerek, kullanırlar.[3] Dolayısıyla yazılım çerçevesinin olabildiğince esnek ve genişletilebilir bir şekilde tasarlanması bir zorunluluktur.[7] Bu özelliklere sahip bir yazılım çerçevesi gerçeklenirken Tasarım Kalıplarından yararlanılması uygun olacaktır. Bu özellikleri sağlamak için, Tasarım Kalıplarını kullanan yazılım çerçevelerinin yüksek seviyede tasarım ve kod tekrar kullanımına ulaşması, Tasarım Kalıplarını kullanmayan çerçevelere göre daha olasıdır. Bu kalıplar, çerçeve mimarisinin, birçok farklı uygulama için tekrar tasarlanmadan kullanılabilmesine yardımcı olur.[7][8]

3. Tasarım Kalıpları

Tasarım, elde edilmek istenen ürünün soyut halidir. Bu soyutlamada ürünü oluşturan parçalar ve bu parçalar arasındaki ilişkiler tanımlanmıştır. Belli bir problem uzayındaki soyutlamaları oluşturan parçalar kavramsal olarak aynıdır. Örneğin, evlerin mimari tasarımları düşünüldüğünde, fiziksel olarak hangi malzemeden ve nasıl yapıldığına bakılmaksızın, her ev için kapı, oda, pencere, mutfak, duvar vb. gibi bileşenler bulunmaktadır. Bunun yanında bir odanın duvarları ve pencereleri içermesi, evin odaları içermesi gibi bileşenler arası ilişkiler de tasarımda yer alır. Aynı şekilde nesneye dayalı tasarmlar da aynı soyutlama özelliklerini gösterir. Nesneye dayalı tasarım çözümlerinde duvarlar ve kapılar yerine nesneler ve arayüzler vardır ancak her iki kalıp çeşidi de temelde, bir problem uzayındaki çözümlerdir.[7] Mimar Christorher Alexander'ın tanımıyla her kalıp, kendi ortamında tekrar tekrar ortaya çıkan problemleri anlatır, sonra, aynı yöntemi tekrar yapmadan, milyonlarca defa kullanılabilecek bir şekilde, bu problemin temel çözümünü ortaya koyar.[7][8]

Her nesneye dayalı yazılım ürününde tekrarlayan problemler vardır. Nesneleri kimin yaratacağı, çalışma zamanında nesnelerin özelliklerinin değişmesi gerekliliği gibi problemler, geliştirilen programlama dili ve ortamdan bağımsız olarak karşımıza çıkar. Bu problemlere tasarım kalıpları aracılığı ile genel çözümler bulunabilmektedir. Nesneye dayalı tasarımlar için kullanılan tasarım kalıpları özel bir bağlamda, genel tasarım problemlerinin çözümlerini ortaya koymamızı sağlayan tanımlamalardır. Bu tanımlamalar ile, tasarım kalıpları tekrar kullanılabilen nesne tabanlı tasarımları ortaya çıkarır çünkü tasarım kalıpları genel tasarım yapısını isimlendirir, soyutlaştırır ve tanımlar. [9]

Sık sık kullanılan birçok tasarım kalıbının belgelendiği klasik kitap Tasarım Kalıpları: Tekrar Kullanılabilir Nesne Tabanlı Yazılım Elemanları kitabında[7] tasarım kalıpları, kalıpların amaçlarına ve uygulama alanlarına göre Yaratımsal, Yapısal ve Davranışsal olmak üzere üç ana başlık altında toplanmıştır. Yaratımsal Tasarım Kalıpları, nesnelerin yaratılma işlemlerini soyutlamak için kullanılmaktadır. Bu kümedeki kalıplar bir sistemin, nesnelerin nasıl yaratıldığınından bağımsız olması amacıyla kullanılmaktadır. Yapısal Tasarım Kalıpları, sınıfların ve nesnelerin daha büyük yapılar oluşturabilmek için nasıl bir araya getirileceği ile ilgili problemlere çözümler sunmaktadır. Davranışsal Tasarım Kalıpları da algoritmalar ve nesneler arasındaki sorumluluk atamaları ile ilgili konularla ilgilenmektedir. [7] [8]

4. Üniversite Finans Projeleri İçin Bir Yazılım Çerçevesi

Tüm üniversitelerin finans birimlerindeki uygulamalar belirli kanunlar ve kurallar dahilinde yapıldığı için bu uygulamaların ortak noktaları oldukça fazladır. Ne yazık ki neredeyse her üniversite kendi finans projesini, en temel adımlardan başlayarak geliştirmekte ve bu durum yazılım mühendisliğinin en temeli olan tekrar kullanımı azaltmakta ve kaynakların aynı işlemler için tekrar tekrar harcanmasına yol açmaktadır. Bu bölümde üniversitelerin finans birimleri için hazırlanacak projelere temel oluşturacak bir çerçevenin bazı bileşenlerinin geliştirilmesinde tasarım kalıplarının, tekrar kullanımı ve genişletilebilirliği desteklemek için, nasıl kullanıldığı anlatılmaktadır.

4.1 Doğrulama/Yetkilendirme

Doğrulama, bir istemcinin kim olduğunun belirlemesidir. Bu işlem normal olarak bir kullanıcı adı ve parola kontrolü ile gerçekleştirilir.[2] Yetkilendirme ise doğrulama aşamasını geçmiş bir kullanıcının sistemde bulunan kaynaklara ulaşabilme seviyesinin belirlenmesidir.[2]

Bir *Windows* uzayında bulunan kullanıcının doğrulama ve yetkilendirme işlemi *Active Directory* Servisi(ADS) ile yapılabilir. Bu durumda ADS, verilen kullanıcı adı ve parola ile kullanıcının kimliği ve dahil olduğu güvenlik gruplarını sistem için hazırlar. Ancak bu işlemlerin her zaman ADS kullanılarak yapılması mümkün olmayabilir. Örneğin, sistem uzayında bir ADS bulunmayabilir, sistem bir *Windows* uzayı içinde olmayabilir ya da özel nedenlerle kullanıcı ad, parola ve rol bilgileri bir veri tabanında saklanıyor olabilir. Bu durumda geliştireceğimiz Doğrulama ve Yetkilendirme Sistemi'nin faklı yapılar üzerinden çalışabilmesi ve gelecekte bu sistemlerin değişmesine olanak vermesi gerekmektedir.

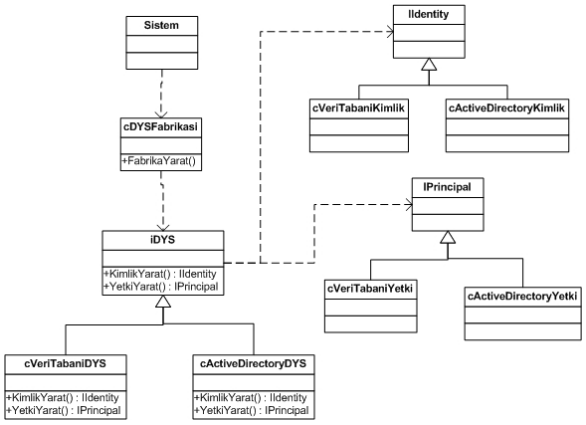
4.1.1 *.NET* ve Doğrulama ve Yetkilendirme İşlemi

.NET uygulamalarında kullanıcı kimliği sunan sınıfların *System.Security.Principal.IIdentity* arayüzünü uygulaması gerekmektedir. Kullanıcı rollerini içeren sınıfların ise kullanması gereken arayüz *System.Security.Principal.IPrincipal'* dır.

4.1.2 Doğrulama ve Yetkilendirme Sistemi (DYS) ve *Abstract Factory* Tasarım Kalıbı

Geliştirilen DYS' yi kullanacak bir uygulama geliştiricinin kullanıcı kimliğinin ve rollerinin hangi sistemler üzerinden belirlendiğini bilmesi, bilmek zorunda olması, geliştiriciye ek bir yük getirecektir. Bu sebeple kullanıcıyı bu işlemden soyutlamak için *Abstact Factory Kalıbı'* ndan yararlanılabilir. *Abstract Factory*, birbirine bağımlı ya da ilişkili nesne gruplarının yaratım işini istemciden soyutlamak için kullanılmaktadır. [7]

DYS'de kullanıcı kimliğinin ve rollerinin belirlenmesi için kullanılabilecek *Active Directory* ve veri tabanı yapılarının bulunduğu varsayılmıştır.



Şekil 1: DYS UML Diyagramı[7]

DYS UML diyagramındaki dikkat edilmesi gereken ilk nokta Sistem'in sadece cDYSFabrikasi ile ilişki içinde olduğudur. cDYSFabrikasi tipinden bir nesne yaratıldıktan sonra bir değişkenden alınan bilgiye göre cVeriTabaniDYS ya da cActiveDirectoryDYS tipinden bir nesne yaratılacaktır. Yaratılan nesne de ilgili *IIdentity* ve *IPrincipal* tipinden nesneleri yaratacaktır.

Abstract Factory Kalıbının kullanıldığı bu tasarımda Sistem, *IIdentity* ve *IPrincipal* tipinden nesnelerin yaratılması ile ilgili ayrıntılarla ilgilenmek zorunda değildir. Sistem kodunun içinde IIdentity ve *IPrincipal* tipinden somut nesneler yaratılmasına gerek yoktur. Yeni bir doğrulama ve yetkilendirme yapısının kullanılması gerektiğinde ise Şekil 1 deki yapı kullanılarak cYeniDYS sınıfının yaratıldığında, Sistem kodunun güncellenmesine gerek kalmayacaktır çünkü Sistem sadece tanımlanmış arayüzlerle çalışmaktadır.

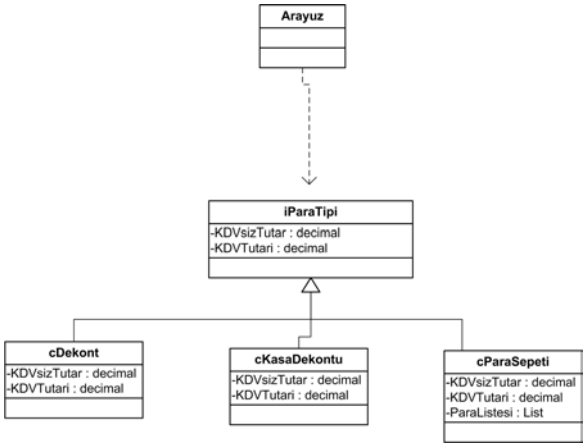
4.2 Sepet Yapısı

Üniversitelerin finans birimleri tarafından kullanılan uygulamalardaki ortak bileşenlerden diğer biri ise para miktarlarını, KDV oranlarını vb. bilgileri içeren banka dekontu, kasaya nakit yatırılan paralar için kasa dekontları ve bağış makbuzları gibi yapılardır. Bu yapılardan bundan sonra para tipleri olarak bahsedilecektir.

Uygulamalarda, bahsedilen para tipleri nesneleri tek tek kullanıldığı gibi farklı para tipleri nesnelerinden oluşan kümeler de kullanılabilmektedir. Örneğin, bir işlem için seçilmiş paraların toplam miktarları gösterilmek istendiğinde, para tiplerine bakılmaksızın tüm paraların miktarlarının toplamı gösterilmelidir. Bu örnekler çoğaltıldığında görülmektedir ki uygulamalar çoğu zaman tek bir para ya da farklı para tiplerin türetilmiş paralardan oluşmuş bir kümeyle uyumlu çalışmalıdır. Bu problemi çözmek için *Composite* Kalıbı'ndan yararlanılabilir.

4.2.1 Sepet Yapısı ve *Composite* Kalıbı

Composite Tasarım Kalıbı, uygulamaların tek bir nesneye ya da nesnelerden oluşmuş kümelerle aynı şekilde çalışmasına yardımcı olmaktadır.[7] Diğer bir ifadeyle seçilmiş tek bir para ya da paraların toplam miktarını ekranda gösteren bir arayüz, farklı para tiplerinden ve para sayısından etkilenmemelidir.



Şekil 2: Sepet UML Diyagramı[7]

Bu tasarımda iSepetParasi arayüzü' nün KDVsizTutar ve KDVTutari isimli 2 özelliği vardır dolayısıla iSepetParasi arayüzünü uygulayan cParaSepetiParasi ve cParaSepeti sınıfları da aynı metodlara sahiptir. Buradaki can alıcı özellik para miktarları bilgilerini ekranda gösterecek olan Arayuz sınıfının sadece iParaTipi arayüzü ile ilişki içinde olmasıdır. Bu sebeple Arayuz sınıfının ekranda göstereceği KDVsizTutar ve KDVTutari bilgilerinin, cParaSepeti, cDekont ya da cKasaDekontu tipi nesnelerden gelmiş olması, Arayuz sınıfı açısından önemli değildir. Böylelikle Arayız sınıfı bir ya da birden fazla farklı para tipi nesneleri için aynı işlemleri yapabilecektir.

İleride farklı para tiplerinin sisteme eklenmesi durumunda, yeni para tipi sınıfı da iParaTipi arayüzünü uygulayacağından, Arayuz nesnesi ile iParaTipi arasındaki ilişkide değişiklik olmayacaktır.

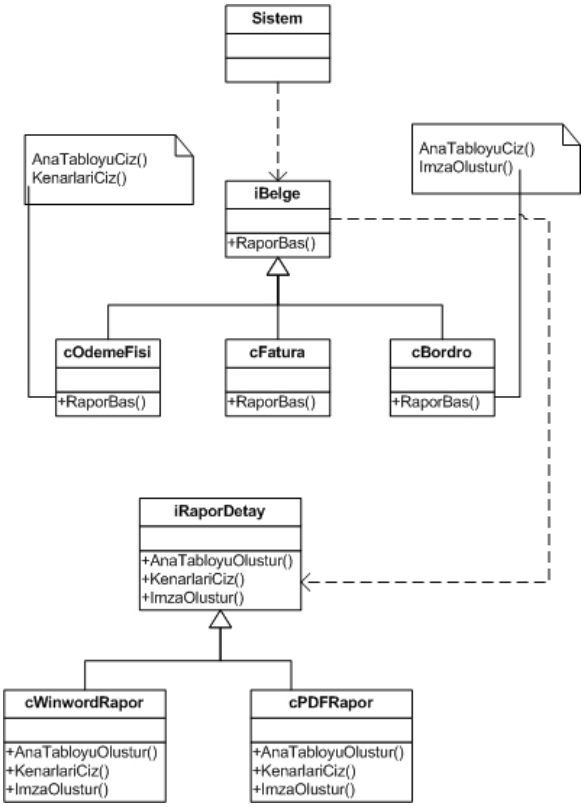
4.3 Raporlama

Üniversitelerin finans birimlerinin kullanacağı uygulamaların temelini oluşturacak bir çerçevedeki bileşenlerden biri de Raporlama Modulü' dür. İşlem sonuçlarını kontrol eden birimlerin isteklerine göre raporların formatları (*Excel*, *Pdf*, *Word* vb.) ve içerdikleri bölümler (tarih, açıklama alanı, imza vb.) farklı olabilmektedir. Bunun yanında rapor formatlarının çalışma zamanında belirlenmesi gibi istekler de karşılanmalıdır. Dolayısıyla geliştirilecek sistemin bu isteklere ve gelecekte olabilecek değişiklik isteklerine cevap verebilecek yapıda tasarlanması gerekmektedir.

4.3.1 Raporlama Yapısı ve *Bridge* Kalıbı

Raporlama yapısındaki temel amaç uygulama geliştiricilerin rapor hazırlarken farklı bölümleri rapora ekleyebilmelerini ve rapor formatlarını istedikleri gibi değiştirebilmelerini sağlamaktır. Farklı bir söyleyişle rapor oluşturulurken kullanılacak bölümleri belirleyen metodların ve bu metodların uygulanışlarının birbirinden ayrılması gerekmektedir. Bu problemi çözmek için *Bridge* Kalıbından yararlanabiliriz.

Bridge Kalıbı, soyutlamanın ve bu soyutlamanın uygulanışlarını birbirinden ayırmak için kullanılır. Böylelikle soyutlama ve uygulama birbirinden bağımsız şekilde değiştirilebilir. [7]



Şekil 3: Raporlama UML Diyagramı[7]

Raporlama yapısında dikkat edilmesi gereken temel nokta Sistem' in sadece iBelge arayüzü ile ilişki içinde olmasıdır. cFatura, cOdemeFisi ve cBordro sınıflarının RaporBas() metodları ve bu metodların raporları farklı formatlarda hazırlamak için kullanacakları farklı bileşenler birbirinden ayrılmıştır. Dolayısıyla bu ayrılmış yapılar birbirlerini etkilemeden değiştirilebilmektedir. Ayrıca yeni bir rapor formatı ya da raporların içeriklerinin değiştirilmesi isteğine istemci etkilenmeden cevap verilebilmektedir.

5. Sonuçlar

Bu çalışmada üniversitelerin finans birimlerinde kullanılacak uygulamalara temel oluşturması için geliştirilen bir çerçevenin bazı bileşenleri anlatılmıştır. Doğrulama ve Yekilendirme Sisteminde *Abstract Factory* Kalıbının, Sepet yapısında *Composite* Kalıbının, Raporlama sisteminde ise *Bridge* Kalıbının bileşenlerinin geliştirilmesinde kullanılması , tekrar kullanılabilir ve gelecekte ortaya çıkabilecek değişiklik isteklerini cevaplayabilen, genişletilebilir tasarımlara ulaşmaya yardımcı olduğu görülmüştür. Ayrıca tasarımlar için gelebilecek değişiklik isteklerine nasıl cevap vereceğinin önceden belirlenebilmesi de tasarım kalıpları kullanımının bir avantajı olarak ortaya çıkmıştır.

6. Kaynaklar

[1] M. Mattsson, "Effort Distribution in a Six Year Industrial Application Framework Project", Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference, Oxford, UK, 1999, pp. 326-333

[2] X. Chen, "*Developing Application Frameworks in .NET*", Apress, USA, 2004.

[3] A. Krajnc, M. Hericko, "Classification of Object-Oriented Frameworks", EUROCON 2003. Computer as a Tool. The IEEE Region 8, 2003, pp. 57-61.

[4] S.F. Lopes, A.C. Tavares, C.A. Silva, J.L. Monteiro, "Application Development by Reusing Object-Oriented Frameworks", Application Development by Reusing Object-Oriented Frameworks", Computer as a Tool, 2005. EUROCON 2005.The International Conference on Volume 1, 2005, pp. 583-586.

[5] M. Morisio, D. Romano, C. Moiso, "Framework Based Software Development: Invesigating the Learning Effect", Software Metrics Symposium, 1999. Proceedings. Sixth International, 1999, pp. 260-268

[6] P.W. Fach, "Design Reuse through Frameworks and Patterns", Software, IEEE Volume 18, Issue 5, 2001, pp. 71-76

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison Wesley Professional, Indianapolis, 1995

[8] Shalloway A., Trott J. R., "Desgin Patterns Explained A New Perpective on Object-Oriented Design", Addison Wesley Professional, 2001

[9] S. Srinivasan, "Design Patterns in Object-Oriented Frameworks", Computer Volume 32, Issue 2, pp. 24-32

*eGİS: Gömülü Sistemler İçin Tasarım Desenleri Tabanlı, Nesneye Yönelik ve Gerçek Zamanlı Bir Mikroçekirdek

K. Sinan YILDIRIM Aylin Kantarcı
Bilgisayar Mühendisliği Bölümü, Ege Üniversitesi, 35100, Bornova, İzmir, Türkiye
sinan.yildirim@mail.ege.edu.tr
aylin.kantarci@ege.edu.tr

Özet

*Bu çalışmada gerçek zamanlı ve gömülü sistemler üzerinde çalışan işletim sistemleri ve bu işletim sistemlerini kullanan gömülü yazılımların yazılımsal özellikleri ve gereksinimleri incelenmiştir. Bu doğrultuda uygulama ihtiyaçlarına yönelik olarak şekillenebilen, taşınabilir, gerçek zamanlı, gömülü ve nesneye yönelik bir işletim sistemi olan **eGe Gömülü İşletim Sistemi** (eGİS) geliştirilmiştir. Bunun yanı sıra, eGİS mimarisindeki esnekliğin ve değişebilirliğin sağlanmasında kullanılan tasarım desenlerinin sisteme kattığı faydalar ve getirdiği yan etkiler tartışılmıştır. eGİS işletim sisteminin var olan diğer gerçek zamanlı ve gömülü işletim sistemleri ile kıyaslaması yapılmış, eGİS'in bu sistemlere olan benzerlikleri ve bu sistemlerden farkları ortaya konulmuştur.*

1. Giriş

Bilgisayar araştırmacıları uzun bir süre gömülü yazılıma olan ilgilerini düşük tutmuştur. Yazılımın küçük olması ve genellikle birleştirici dili ile yazılması; ayrıca sistemin donanım maliyetinin yazılım maliyetine göre fazla oluşu, gömülü sistemler için yazılım geliştirmenin temel sorunları olmuştur. Yazılım mühendisliğinin sunduğu çözümlerin gömülü sistemler için çok pahalı görünmesi ve bu alana uygulanabilir olmaması, gömülü sistemlerdeki yazılımsal çalışmaları ikinci plana atmıştır. Ancak, günümüzde donanımın daha ucuz hale gelmesi, buna ek olarak yazılımın daha büyük ve karmaşık bir hal alması nedeniyle gömülü sistemlerdeki yazılım araştırmaları tekrar popüler hale gelmiştir [1].

Günümüz gerçek zamanlı sistemleri giderek daha büyük ve karmaşık yazılımlar içermektedirler. Artık gömülü sistemler birleştirici dili ile kolayca yazılabilen ve tasarlanabilen yazılım sistemleri değildirler. Yeni mimarilere taşınabilme, değişen piyasa koşullarına göre alınan değişim isteklerine duyarlı olabilme ve çok değişken gereksinimlere sahip sistemlerde de yeniden kullanılabilme gibi özellikler gömülü yazılımlarda da modern yazılım mühendisliğinin ortaya koymuş olduğu kavramların göz önünde bulundurulmasını gerektirmektedir.

Modern gömülü yazılım sistemlerinin sahip olması gereken temel özellikler şu sekilde sıralanabilir [2] :

- Güçlü bir yazılım mimarisi
- Sistem bileşenlerinin yeniden kullanılabilir olması
- Dağıtıklık (Alt sistemlerin dağıtılabilir olarak tasarlanması)
- Ortamdan bağımsız olma ve değişik sistem mimarilerine ve donanımsal ortamlara taşınabilme
- Genişleyebilir ve isteğe göre değiştirilebilir alt sistemlere sahip olma ve uygulama ihtiyaçlarına göre yapılandırılabilme
- Kaynakların verimli kullanılması

Gömülü sistem yazılımlarının yukarıda belirtilen özelliklere sahip olması, kendini ispatlamış ve başarıya ulaşmış kaliteli tasarımların yeniden kullanılmasına dayanan tasarım desenlerinin [3] bu sistemlere de uygulanması ile gerçekleştirilebilir. Mevcut başarıya ulaşmış tasarımları kullanmak, uygulamaya, alanına özel problemlerin çözümü için iyi bir başlangıç yapmayı sağlar ve ortaya çıkabilecek hataları engeller.

Tasarım desenlerinin bir sistemde etkileyebileceği noktalar şu şekilde sıralanabilir [4]:

- Sistem başarımı
- Tahmin edilebilirlik
- Planlanabilirlik
- Verim
- Güvenilirlik

* Bu çalışma TÜBİTAK-BAYG (Türkiye Bilimsel ve Teknolojik Araştırma Kurumu) tarafından desteklenmiştir.

- Yeniden kullanılabilirlik
- Dağıtıklık
- Taşınabilirlik
- Bakım
- Genişleyebilirlik
- Karmaşıklık
- Kaynak kullanımı
- Enerji tüketimi
- Donanım maliyeti
- Sistem geliştirme emeği ve maliyeti

Görüldüğü gibi tasarım desenlerinin sunduğu çözüm yolları, getirdikleri avantajların yanında, özellikle gömülü sistemler için oldukça önemli noktalarda sistemi olumsuz yönde etkileyebilmektedir. Örneğin, tasarım desenlerinin kullanılması ile ortaya çıkan başarım kaybı, sistem içerisindeki görevlerin belirlenmiş zaman kısıtı altında bitirilmesini engelleyebilir. Gerçek zamanlı sistemlerin tahmin edilebilirlik özelliği, tasarım deseni kullanmanın getirmiş olduğu ek soyutlama katmanlarından dolaylı olarak etkilenebilir. Dolayısıyla gömülü bir sistem tasarlanırken o sistemin gereksinimlerini tam ve eksiksiz bir şekilde yerine getirmesi için, tasarım desenlerinin getirdiği iyileştirmeler ve ek yükler dikkatli bir şekilde göz önüne alınmalıdır.

Gömülü sistemler için gerçekleştirilen işletim sistemleri, gömülü sistemler üzerinde çalışacak diğer yazılımlar gibi, genel amaçlı işletim sistemlerinden farklı tasarım amaçlarına ve servislere sahiptirler. Verimli bir kaynak yönetimini ve gerçek zamanlı sistem algoritmaları içeren gömülü işletim sistemleri, giderek modern yazılım kavramlarının sunmuş olduğu avantajlara gereksinim duymaktadır. Gömülü işletim sistemi kullanıcıları ve geliştiricileri, daha genel ve değişik istekleri daha rahat karşılayacak bir mimariye sahip sistemlere gün geçtikce daha fazla ihtiyaç duymaktadırlar. Özellikle gerçek zamanlı sistemlerin farklı ortamlarda çalışma gerekliliği ve çok değişik istek ve kısıtlara sahip olduğu düşünülürse, genel ve bu isteklere göre değiştirilip uyarlanabilecek bir işletim sistemi mimarisi büyük bir ihtiyaç olarak ön plana çıkmaktadır.

Gömülü işletim sistemlerinde tasarım desenlerinin kullanılması, işletim sisteminin uygulama gereksinimlerine göre değiştirilip yeniden yapılandırılmasını kolaylaştırır. Uygulama gereksinimleri gömülü sistemlerde derleme anında belirli olduğu için, burada tasarım desenlerinin çalışma zamanında sunduğu değişebilirlik değil; derleme zamanında sunduğu değişebilirlik daha çok ön plandadır. İşletim sistemi derleme zamanında, uygulama gereksinimlerini karşılayan alt sistemleri ile derlenir ve alt sistemlerin kendi aralarında değişebilirliğinin sağlanmasında tasarım desenleri önemli yer tutar.

Yeniden kullanılabilirlik, işletim sistemleri ve gömülü yazılımlar için de çok önemli bir noktadır. Hata bulma ve ayıklama işlemlerinin diğer yazılım sistemlerine göre daha zahmetli olduğu düşünülürse, aynı kodun tekrar tekrar yazılıp hata ayıklama işlemlerinin yapılması yeniden kullanılabilirlik ile engellenecektir. Ayrıca işletim sistemi kod büyüklüğünün düşmesi gömülü sistemler için önemlidir. İyi test edilmiş ve verimlilik göz önüne alınarak gerçekleştirilmiş alt sistemlerin yeniden kullanılması ile, kod büyüklüğü azaltılabilir.

Bir işletim sisteminin tasarımı sırasında, diğer bir önemli nokta ise monolitik ve mikroçekirdek mimariler arasında doğru bir seçim yapabilmektir.

Monolitik [5] işletim sistemleri modüler olarak tasarlanmışlardır. İşletim sistemi çekirdeği sistemdeki gerekli tüm önemli servisleri sunmaktadır. Bellek yönetimi, süreç yönetimi, giriş/çıkış yönetimi gibi işlevlerin tamamı çekirdeğin içerisinde yönetilmektedir. Monolitik sistemler modüler yapıya sahip olmalarına rağmen, genişleyebilirlik ve dağıtıklık özelliklerinin bu sistem mimarisi içerisinde gerçeklenmesi daha zordur.

Mikroçekirdek [5] mimarisi ise, modern ve yazılım mühendisliğine daha yatkın bir işletim sistemi mimarisidir. Burada amaç minimal bir çekirdek sağlamaktır. Küçük bir mikroçekirdek ile daha temiz bir arayüz sunulur ve bu sayede daha modüler bir sistem elde edilir. Ayrıca küçük olan çekirdek ile bakım işlemi kolaylaşır ve hatalara karşı duyarlılık artar. Ayrıca mikroçekirdeğin servislerinden faydalanan sunucular ile dağıtıklık ve sistemin taşınabilirliği artacaktır.

Bu çalışma kapsamında, gerçek zamanlı gömülü uygulamaların gereksinimlerini karşılayacak olan eGe Gömülü İşletim Sistemi (eGİS) [6], C++ dili ve açık kaynak koda sahip ücretsiz yazılım geliştirme araçları olan Gcc derleyicisi, Gdb hata ayıklayıcısı, Ld bağlayıcısı ve Bochs emülatörü kullanılarak geliştirilmiştir. eGİS'in tasarımında modern yazılım kavramları ve özellikle tasarım desenlerini taban alan mikroçekirdek mimarisi göz önüne alınmıştır. eGİS bir uygulamaya yönelik işletim sistemi olarak uygulama ihtiyaçlarına göre değiştirilebilir ve biçimlendirilebilir bir yapıya sahiptir. eGİS'in bir başka özelliği ise tamamiyle Türkçe gerçekleştirime sahip olmasıdır.

Bildirinin geri kalan kısmı şu şekilde düzenlenmiştir: İkinci bölümde eGİS yaygın olarak

kullanılan ticari ya da açık kaynak kodlu diğer gömülü işletim sistemleri ile karşılaştırılmış, benzerlikler ve farklılıklar belirtilmiştir. Üçüncü bölümde eGİS'in mimarisi ve tasarım hedefleri açıklanmıştır. Dördüncü bölümde tasarım desenlerinin eGİS işletim sisteminde uygulandığı noktalar belirtilmiş; bunların sisteme kattığı faydalar açıklanmıştır. Beşinci bölümde ise eGİS'in Bochs emülatörü üzerinde çalıştırılması sonucunda elde edilen deneysel veriler kullanılarak tasarım desenlerinin sisteme getirdiği ek yükler incelenmiştir. Altıncı bölümde, çalışma özetlenerek gelecekte yapılabilecek araştırma ve geliştirmeler tartışılmıştır.

2. İlgili Çalışmalar

eGİS işletim sistemi, Linux ve Windows gibi genel amaçlı işletim sistemlerinden hem mimarisel hem de amaç olarak ayrılmaktadır. Linux ve Windows gibi masaüstü uygulamalarına yönelik olan işletim sistemleri, gerçek zaman kısıtları düşünülerek gerçekleştirilmiş işletim sistemleri değildir. İçerdikleri süreç yönetim algoritmaları gerçek zamanlı sistemler için uygun değildir [7]. Ayrıca hem Windows hem de Linux monolitik bir modüler yapıya sahiptirler. İşletim sisteminin genişlemesi ve eklentileri Linux'te dinamik olarak eklenebilen modüllerle, Windows'da ise DLL'ler ile yapılmaktadır. Windows'da bütün kullanıcı arayüzü işletim sistemine gömülmüştür. Her iki işletim sistemi de bellek problemi olmayan çok geniş sistemler için tasarlanmıştır. eGİS ise gömülü bir mikroçekirdektir. Gerçek zaman kısıtlarını göz önüne alan yönetim algoritmalarını içermektedir.

uCOS-II [8] işletim sistemi, gömülü gerçek zamanlı sistemler için gerçekleştirilmiş ticari bir işletim sistemidir. C dili ile gerçeklenen uCOS-II, monolitik bir yapıya sahiptir. Tahmin edilebilir algoritmalara sahip olan uCOS-II, modern yazılım kavramları göz önüne alınarak gerçekleştirilmiş bir işletim sistemi değildir. Öncelik tabanlı kesilebilir bir süreç yönetim algoritması kullanan uCOS-II, bu algoritmanın değişimi için bir mekanizma sağlamamaktadır. Bu algoritmanın değişimi, işletim sisteminin önemli bir bölümünün yeniden yazılmasını gerektirir. Küçük bir kod büyüklüğüne sahip olan uCOS-II, oldukça iyi sistem başarımına sahiptir. Ancak uygulamanın ihtiyaçlarına göre şekillenebilen bir sistem olmadığı için, değişimlere duyarlı değildir. eGİS uCOS-II ile kıyaslandığında, daha düşük fakat kabul edilebilir bir başarıma sahip olsa da, daha yüksek bir değişebilirliğe sahiptir. Mikroçekirdek mimarisi sayesinde eGİS, uCOS-II'nin sahip olmadığı genişleyebilirlik ve donanıma daha rahat taşınabilirlik özelliklerini içermektedir.

Nucleus [9] gömülü gerçek zamanlı işletim sistemi de, uCOS-II gibi ticari bir işletim sistemidir. Nucleus işletim sistemi de C dili ile gerçekleştirilmiş olup moduler yapıya sahip bir mimariye sahiptir. Ancak Nucleus da uCOS-II gibi modern yazılım kavramları göz önüne alınarak gerçekleştirilmiş bir işletim sistemi değildir. Nesneye yönelik tasarlanmamış olan Nucleus işletim sistemi, değişimlere eGİS gibi duyarlı değildir. Nucleus işletim sistemi de, belirli bir süreç yönetim algoritmasına bağlıdır. Bu algoritmanın değişebilirliği sistem içerisinde sağlanmamıştır. İşletim sisteminin donanım bağımsızlığı da bir ek katman ile sağlanmıştır. Bu ek katman eGİS sistemindeki köprü deseni ile kıyaslandığında çok esnek değildir.

eCOS [10] işletim sistemi, C++ dili kullanılarak nesneye yönelik olarak geliştirilmiş açık kaynak koda sahip bir işletim sistemidir. eCOS'un ana amacı uygulama ihtiyaçlarına göre şekillenebilen bir işletim sistemi mimarisidir. Ancak eCOS düzenli bir nesneye yönelik ayrıştırım içermemekte ve genellikle sınıf kalıtımına dayanmaktadır. eCOS'un sınıf hiyerarşisi ve mimarisi, tasarım deseni tabanlı değildir. Bir çok mimariye taşınmış olan ve birçok bileşen içeren eCOS'un nesneye yönelik mimarisi katmanlı bir modüler yapıya benzemektedir. eGİS ise mikroçekirdek mimarisini tasarım desenlerinin getirdiği esneklik ile değişimlere açık hale getirmiştir.

PURE [11] işletim sistemi gömülü sistemler için geliştirilmiş ve uygulamaya yönelik bir işletim sistemidir. PURE sınıf hiyerarşisi ve sınıf kalıtımına dayalı bir sistem mimarisine sahiptir. eGİS ile kıyaslandığında, çok geniş ve karmaşık bir sınıf hiyerarşisine dayanır. eGİS sistemi ise tasarım desenlerinin de üzerine oturduğu arayüz kalıtımına dayanan bir işletim sistemidir.

CHORUS [12] mikroçekirdek mimarisine sahip bir işletim sistemidir. C++ dili ile gerçekleştirilmiştir ve bir sınıf kalıtımı ve hiyerarşisine dayanmaktadır. Ayrıca CHORUS bileşenleri çok büyük ve uygulama amaçlarına göre değiştirilebilir değildir. eGİS ise küçük ve bağımsız bileşenlere sahiptir ve arayüz kalıtımına dayanmaktadır. Ayrıca, eGİS'ten farklı olarak, CHORUS tasarım deseni tabanlı bir sistem değildir.

TINYOS [13] işletim sistemi bileşen tabanlı bir işletim sistemidir. Tasarım desenleri bu işletim sistemi içerisinde de yer almaktadır. Ancak TINYOS işletim sisteminde kullanılan desenler bileşen tabanlı desenlerdir ve bu bileşenler derleme anında sistem ile birleştirilmesi işlemlerini yerine getirmek için kullanılırlar. TINYOS, kendi sistemine özgü nesC dili

ile, var olan bileşenlerin uygulama gereksinimlerine göre birbirlerine bağlanmasını tasarım desenleri ile sağlar. İşletim sistemini kullanacak olan uygulamanın gereksinimlerine göre seçilen TINYOS işletim sistemi bileşenleri, tasarım desenlerinin derleme anında sunmuş olduğu değişebilirlik özelliği ile sistemle birleştirilmektedir. eGİS'te ise şu anda işletim sistemi yapılandırılması herhangi bir üst seviye dille değil elle yapılmaktadır. Dolayısıyla eGİS'teki tasarım desenleri bileşenlerin birbirleri ile bağlanmasından daha çok sistem içerisinde yer alan alt sistemlerin tasarımları ile ilgilidir. Ayrıca eGİS'te de tıpkı TINYOS'taki gibi desenlerin derleme anında sunduğu avantajlardan faydalanılmıştır.

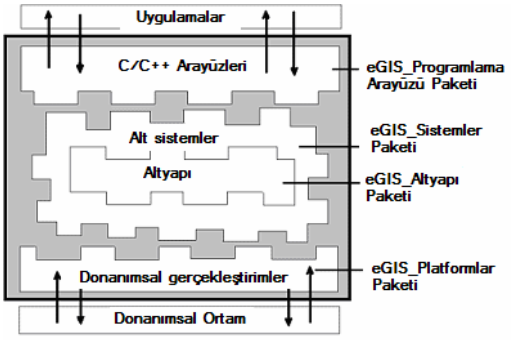
EPOS [14] işletim sisteminin ana amacı işletim sistemini kolaylıkla uygulama programının istediği yapıya kavuşturmaktır. Bu işletim sisteminde de arayüz ve gerçekleştirim ayırımı ile yapılır. EPOS'ta sistem derleme anında kullanıcıdan elde edilen bilgiler doğrultusunda ilgili alt sistemlerle derlenir. Burada kullanıcı bilgisi sayesinde sisteme ek yük getiren soyutlamalardan derleme anında kurtulunur. eGİS'te EPOS'taki gibi bir çalışma henüz yapılmamıştır. Sistem mümkün olduğunca sanal fonksiyon ve dinamik bellek tahsisinden kaçınsa da, henüz desenlerin kullanımı ile oluşan ek yük EPOS'taki gibi giderilememiştir.

3. eGİS Sistem Mimarisi

eGİS gerçek zaman kısıtlarına sahip gömülü sistemler için, genişleyebilir, değiştirilebilir ve taşınabilir bir nesneye yönelik mimariye sahip olacak şekilde tasarlanmış ve tasarım desenleri kullanılarak gerçekleştirilmiş bir mikroçekirdektir. eGİS süreç yönetimi, kesme yönetimi ve süreçler arası haberleşme işlevlerini yerine getiren üç alt sistemden oluşmaktadır.

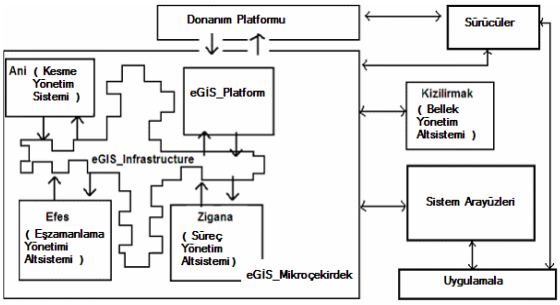
eGİS'in katmanlı mimarisi Şekil 1'de gösterilmiştir. eGIS *C++ sistem arayüzleri*, *alt sistemler*, *altyapı* ve *platform* katmanlarını içermektedir. Sistemdeki temel arayüzler ve soyutlamalar altyapı katmanında tanımlanmakta ve altsistemler katmanında bu soyutlamaların gerçekleştirimleri yer almaktadır. Örneğin temel süreç soyutlamaları altyapı katmanında yer alırken, uygulama gereksinimlerine göre süreç gerçekleştirimi altsistemler katmanında yer almaktadır. Platform katmanı donanıma özgü gerçekleştirimleri içermektedir. Altsistemler ve altyapı katmanları bu sayede hiçbir donanımsal bağımlılık içermezler.

C++ arayüzleri katmanı uygulamalar için bir uygulama geliştirme arayüzü sunmaktadır. Bu sayede uygulamaların katmanlı mimariden ve sistemin iç yapısından haberdar olmaları engellenmiştir. Ayrıca alt katmanlarda meydana gelen herhangi bir değişiklik, uygulama katmanında bir değişikliğe neden olmayacaktır.



Şekil 1. eGİS'in katmanlı mimarisi

eGİS mikroçekirdeğinin içsel yapısı Şekil 2'de gösterilmiştir. Kesme, süreç ve haberleşme sistemleri sırası ile ***Ani***, ***Zigana*** ve ***Efes*** alt sistemlerinde gerçekleştirilmişlerdir. *Zigana* öncelik tabanlı kesilebilir süreç yönetim algoritmasının gerçekleştirildiği alt sistemdir. *Zigana* süreç yaratma, öldürme, durdurma gibi servisleri sunmaktadır. *Ani* basit ve verimli bir kesme sistemi olup, kesme servislerine kayıt olmak ve bu servisleri işleyebilmek için basit bir mekanizma sağlamaktadır. *Efes* ise süreçler arası haberleşme ve eşzamanlamayı sağlayan semafor/kilit sinyalleme ve serbest bırakma mekanizmalarının gerçekleştirimlerini barındırır.



Şekil 2. eGİS mikroçekirdeğinin iç yapısı ve dış bileşenlerle olan etkileşimi

Şekil 2'de görüldüğü gibi, eGİS içerisinde yer alan tüm alt sistemler donanımdan soyutlanmışlardır. eGİS mikroçekirdeğinin değişik bir donanımsal ortama taşınmasında, mikroçekirdeğin mekanizmalarını kullanan alt sistemler bundan etkilenmezler. Mikroçekirdeği yeni bir donanım ortamına taşımak, sadece donanımsal bağımlılık içeren kısımlarının

değişmesini gerektirir. *eGIS_Platformlar* paketi, *eGIS_AltYapı* içerisindeki donanımsal soyutlamalarının gerçekleştirimlerini içermektedir. Dolayısıyla taşınan ortama ilişkin gerçekleştirimler bu pakete eklenirse, eGİS mikroçekirdeği yeni ortama taşınmış olur. Hiçbir alt sistem bundan etkilenmez. Bu sayede sistem gerçekleştirimi sırasında, tüm alt sistemler kişisel bilgisayar üzerinde birbirlerinden bağımsız olarak test edilebilmişlerdir.

eGİS işletim sisteminin genişleyebilen ve uygulama gereksinimlerine göre değişebilen bir sistem olmasında, arayüz kalıtımı ve nesne içermeye dayalı bir mikroçekirdek mimarisine sahip olmasının rolü büyüktür. İşletim sistemine yeni bir özellik eklemek, mikroçekirdeğe yeni bir alt sistem eklenmesi ile sağlanmış olur. Aynı arayüzleri ancak farklı gerçekleştirimleri içeren alt sistemler, birbirleri ile değiştirilebilirler. Bu sayede, mikroçekirdek tarafından arayüzleri belirlenmiş ama gerçekleştirimleri farklı çok sayıda alt sistem eGİS sistemine uygulama ihtiyaçlarına göre eklenebilir. Yeni bir özelliğin eklenmesi, yeni bir alt sistemin gerçeklestirilmesi ve çekirdeğe kayıt edilmesi ile sağlanır. Örneğin öncelik tabanlı kesilebilir süreç algoritmasının gerçekleştirilmiş olduğu Zigana süreç sistemi, Round-Robin süreç yönetim algoritmasını gerçekleştiren başka bir sistemle değiştirilebilir. Arayüzler ve gerçekleştirimlerin bir işletim sistemi içerisinde ayrıştırılmış olması, eGİS sisteminin esnekliğini güçlendirmiştir.

eGİS işletim sisteminin katmanlı mimarisi özel olarak taşınabilirlik, genişleyebilirlik, bakım ve test edilebilirlik bakımından genel sisteme katkıda bulunmuştur. Ayrıca eGİS'in uygulama ihtiyaçlarına göre şekillenebilmesi sistem gerçekleştiriminde tasarım desenlerinin uygulanması ile kolaylaşmıştır. Desenler, eGİS'in uygulama ihtiyaçlarına göre değişebilecek noktalara uygulanmıştır ve bu noktalardaki değişimler; desenlerin uygulanması sayesinde sistem içerisinde sadece belirli kısımların değiştirilmesini gerektirir.

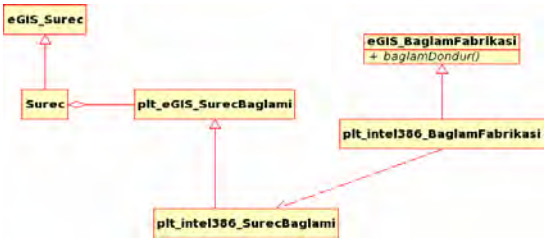
4. Tasarım Desenlerinin eGİS İşletim Sisteminde Uygulanışı

eGİS işletim sisteminde kullanılan tasarım desenleri *yaratımsal*, *yapısal* ve *davranışsal* tasarım desenleri olarak sınıflandırılabilir.

Tekil [3] ve ***Soyut Fabrika*** [3] desenleri, eGİS sistemi içerisinde kullanılan ***yaratımsal*** desenlerdir. Bu desenler sınıfları nesne yaratma işlemlerini soyutlarlar ve sistemin nesnelerin içsel yapıları ve gerçekleştirimlerinden bağımsız hale gelmesini sağlarlar. Sınıfların somut isimlerini açık bir şekilde kullanarak nesneleri yaratmak, sistemin değişebilirliğini ve uygulama gereksinimlerine göre şekillendirilebilir olmasını engellemektedir. Bunun önüne geçmek için, sistem içerisindeki nesne yaratımından dolayı doğacak olan bağımlılıkların ortadan kaldırılması gerekmektedir.

Tekil tasarım deseni ile sınıfların tek bir örneğinin olması garanti altına alınmıştır ve istemciler bu tek örneğin yaratılmasından soyutlanmışlarıdır. eGİS sistemi içerisinde yer alan mikroçekirdek sınıf tekil tasarım deseninin kullanılmasına bir örnektir. Alt sistemler kendilerini mikroçekirdek nesnesine kayıt ederler. Bundan ötürü mikroçekirdek nesnesi kontrollü bir erişimin olması gereken ve istemcilerin bu nesnenin yaratımından bağımsız hale getirilmesini gereken bir sistem elemanıdır.

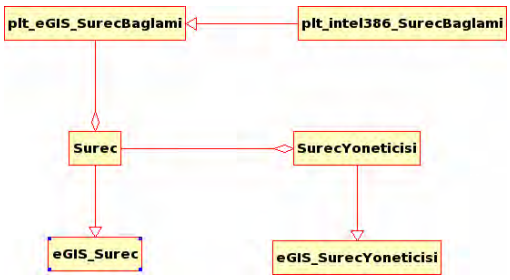
Soyut fabrika deseni, eGİS sistemi içerisinde yer alan nesneleri somut sınıflarını belirtmeden yaratmak için kullanılmıştır. Bu desenin eGİS içerisindeki örnek kullanımı Şekil 3'te gösterilmiştir. Süreç bağlamı ve süreçler arası geçişi sağlayan mekanizmalar, *plt_eGİS_SüreçBağlamı* arayüzünden türeyen sınıflar tarafından gerçekleştirilirler. *eGİS_Süreç* arayüzünden türeyen sınıflar sadece *plt_eGIS_SüreçBağlamı* arayüzünden haberdarlardır. Donanıma özgü alt seviye kodlar platforma özgü sınıflara gömülmüştür. Ancak, donanıma özgü kodlar içeren bu sınıfların sistem içerisinde bir şekilde yaratılmaları ve süreçlere bağlanmaları gerekmektedir. Eğer donanıma bağımlı bu sınıfların yaratımı *eGİS_Süreç* arayüzünden türeyen sınıflar tarafından yapılırsa, sistemde donanımsal kodlar içeren bu sınıflara bir bağımlılık ortaya çıkmaktadır. Sınıflar arasındaki bu bağımlılığı ortadan kaldırmak için platform bağımlı sınıflar *eGIS_BağlamFabrikası* arayüzünü gerçekleştiren sınıflar tarafından yaratılırlar. Sonuç olarak eGİS sistemi donanıma bağımlı olan somut sınıflardan bağımsız hale gelir. eGİS sistemi içerisinde yer alan istemci nesneler somut platform nesnelerinden haberdar olmadan ve sadece onların arayüzlerini kullanarak ilgili isteklerini yerine getirirler.



Şekil 3. Soyut Fabrika desenin eGİS içerisinde kullanımı

eGİS sistemi içerisinde kullanılan yapısal desenler sınıfların birleştirilerek daha büyük işlevsel sınıfların oluşturulmasını ve nesnelerin içerilmesi ile ek işlevsellik kazanılmasını sağlar. Bu sayede eGİS yeni donanımsal ortamlara daha rahat taşınır ve uygulama gereksinimlerine göre daha kolay genişleyebilir. Ayrıca nesneler arasındaki bağımlılıklar da ortadan kalkmış olur. eGİS işletim sistemi içerisinde kullanılan temel yapısal desenler ***Köprü*** [3] ve ***Önyüz*** [3] desenleridir.

Köprü deseni donanım bağımlı işlemleri soyutlar ve bu işlemlere ait arayüzler ve gerçekleştirimlerin birbirlerinden ayrılmasını sağlar. Şekil 4'te nesne içerme mekanizmasına dayanan bu desenin eGİS sistemi içerisindeki kullanımı örneklenmiştir. *plt_eGİS_SüreçBağlamı* arayüzünü gerçekleştiren sınıflar donanım ortamına bağlı yazmaç bilgilerini tutan ve süreçler arası geçişi sağlayan gerçekleştirimleri içerirler. *eGİS_Süreç* arayüzünü gerçekleştiren sınıflar *plt_eGİS_SüreçBağlamı* arayüzünü gerçekleştiren bu nesneleri içererek isteklerini bu nesnelerin sadece arayüzlerini kullanarak gerçekleştirirler. Bu sayede donanım ortamına olan bağımlılıkları ortadan kalkar. Süreç nesnelerinin içerdiği bağlam nesneleri Intel386 donanımsal ortamına ilişkin bir gerçekleştirime sahip olabileceği gibi Mips mimarisine sahip bir donanımsal ortama ait gerçekleştirimleri de barındırabilir. Sonuç olarak, süreç nesneleri donanımsal bağımlılıklar soyutlanmış ve bu da eGİS'in taşınabilirliğini arttırmıştır. eGİS'i başka bir donanım ortamına taşımak için, donanım ortamına uygun bir şekilde *plt_eGİS_SüreçBağlamı* arayüzünü gerçekleştiren sınıfların yazılması ve bunların süreç nesneleri ile ilişkilendirilmeleri gerekmektedir. Bunun dışında işletim sisteminin diğer taraflarında herhangi bir değişime gerek yoktur.

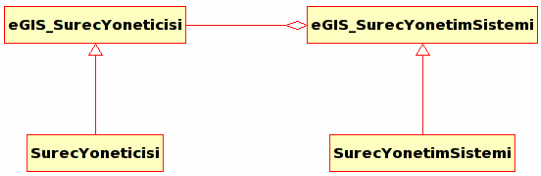


Şekil 4. Köprü desenin eGİS içerisinde kullanımı

eGİS işletim sistemi birçok sınıf ve arayüzleri içerisinde barındırmaktadır. Ancak eGİS'in sunduğu servislerden faydalanmak isteyen istemciler, içsel nesnelerden ve karmaşık sistem yapısından soyutlanmalıdırlar. Sistemin genel, basit ve temiz bir arayüzünün olması gerekmektedir. *Önyüz* tasarım deseni istemcilere üst seviye ve basit bir arayüz sunmaktadır. Şekil 5'te Önyüz deseninin gömülü sistem uygulamalarına bir sistem arayüzünü nasıl sunduğu gösterilmektedir. Üst seviye bir arayüz sunan *eGİS_SistemArayüzü* sınıfı, işletim sisteminin içsel yapısını ve sınıflarını uygulamalardan saklar. Sistem arayüzünü kullanan istemciler sadece *eGIS_SistemArayüzü* 'nden haberdardırlar. Bunlara ek olarak istemciler isteklerini daha az nesne ile haberleşerek gerçekleştirirler.

Şekil 5. Önyüz desenin eGİS içerisinde kullanımı

Davranışsal desenler eGİS sisteminin algoritmik bağımlılıklarını ortadan kaldırmak için kullanılmıştır. ***Strateji*** [3], eGİS içerisinde kullanılan temel davranışsal desendir. Farklı sistem yönetim algoritmaları strateji deseni sayesinde sisteme kolayca eklenebilir. Bu sistem içerisindeki yönetim algoritmalarının uygulama ihtiyaçlarına uygun bir şekilde seçilebilmesini ve sistemin uygulama ihtiyaçlarına göre şekillendirilebilir olmasını sağlar. Algoritma arayüzünü gerçekleştiren uygun bir sınıf, işletim sistemi kodu ile sorunsuz bir şekilde bağlanabilir. Şekil 6 strateji deseninin kullanılması ile eGİS sisteminin kazanmış olduğu esnekliği göstermektedir. eGİS sistemi içerisinde yer alan her süreç yöneticisi, *eGİS_SüreçYöneticisi* arayüzünü gerçekleştirmek zorundadır. Süreç yöneyim servislerini kullanmak isteyen istemciler sadece *eGİS_SüreçYöneticisi* arayüzünü bilmektedirler. Bu sayede algoritmik gerçekleştirimlere olan bağımlılık ortadan kaldırılmıştır. Farklı süreç yönetim algoritmaları yeni gerçekleştirimleri barındıran sınıfların sisteme eklenmesi ile işlevsel hale gelmiş olur. Sistemin diğer kısımlarında herhangi bir değişiklik yapmaya gerek yoktur. Ayrıca algoritmik arayüzler ve gerçekleştirimlerin birbirinden ayrılması yeniden kullanılabilirliği arttırmaktadır.



Şekil 6. Strateji desenin eGİS içerisinde kullanımı

Özetleyecek olursak, eGİS sistemi içerisinde kullanılan desenler işletim sistemi mimarisini daha genişleyebilir ve değiştirilebilir kılmıştır. Bunun sonucunda eGİS, uygulama gereksinimlerine göre şekillendirilebilir ve değişik donanım ortamlarına kolay bir şekilde taşınabilir hale gelmiştir. Donanımdaki ilerlemeler ve gömülü sistemlerde kullanılan işlemcilerin daha da güçlendiği düşünülürse, desenlerin sisteme getirdiği ek yükler giderek daha önemsiz hale gelecektir.

5. Gerçekleştirim ve Deneysel Sonuçlar

eGIS C++ dili ile Gcc [15] C++ derleyicisi, Gdb [16] hata ayıklayıcısı ve Bochs [17] i386 PC emulatörü gibi açık kaynak koda sahip araçlar kullanılarak geliştirilmiştir. Şu anda eGİS sadece i386 ortamına taşınmıştır ve 3 alt sistem ile yaklaşık 60 adet sınıftan oluşmaktadır.

```
plt_intel386_BaglamFabrikasi baglamFabrikasi;
SurecYoneticisi surecYoneticisi;
SurecYonetimSistemi surecYonetimSistemi;

KesmeYonetimSistemi kesmeYonetimSistemi;
KesmeTablosu kesmeTablosu;
plt_intel386_KesmeSistemi plt_KesmeSistemi;

/* baslangic mesajlari */
cout<<"eGIS Isletim Sistemi ver 1.0...\n";
cout<<"Sistem ilkleniyor...\n";

/* surec yonetim alt sistemi olan Zigana kaydediliyor */
surecYoneticisi.platformAta(&baglamFabrikasi);
surecYonetimSistemi.surecYoneticisiAta(&surecYoneticisi);
eGIS_CEKIRDEK->surecYonetimSistemiAta(&surecYonetimSistemi);
cout<<"Surec yonetim sistemi ilklendi...\n";

/* kesme yonetim sistemini ata */
kesmeYonetimSistemi.kesmeTablosuAta(&kesmeTablosu);
kesmeYonetimSistemi.platformAta(&plt_KesmeSistemi);
eGIS_CEKIRDEK->kesmeYonetimSistemiAta(&kesmeYonetimSistemi);
cout<<"Kesme yonetim sistemi ilklendi...\n";

kesmeYonetimSistemi.sistemIlkle();
kesmeYonetimSistemi.kilitle(true);
cout<<"Kesmeler ilklendi...\n";

surecYonetimSistemi.sistemIlkle();
cout<<"Surec yonetim sistemi ilklendi ...\n";

/* bos durum surecini yarat */
bosDurumSureciYarat();
cout<<"Bos surec yaratildi...\n";

/* uygulama surecini yarat */
uygulamaSureciYarat();
cout<<"Uygulama sureci yaratildi ...\n";
```

Şekil 7. eGİS sistem giriş noktası

eGİS'in ana giriş noktası Şekil 7'de gösterilmiştir. Süreç, kesme ve haberleşme yönetimi servislerini sunan alt sistemler sırası ile mikroçekirdeğe kaydedilmekte ve ilgili donanımsal ortam bu sistemler ile ilişkilendirilmektedir. Görüldüğü gibi, desen tabanlı mikroçekirdek sayesinde eGİS'in uygulama isteklerine göre şekillendirilebilmesi kolaydır. Sistem başlangıcı esnasında ilgili alt sistemlerin seçimi ile eGİS yeniden yapılandırılır ve sistemin diğer taraflarında başka bir değişiklik yapılmasına kalmaz. eGİS'i yeni bir donanım ortamına taşımak, ortama özgü gerçekleştirimleri barındıran nesnelerin sisteme kaydedilmesi ile olmaktadır.

```
/*
 * Sureclerin yaratilmasi ve surec yonetim
 * isteklerinin ilgili yoneticilere aktarilmasi
 * islemleri icin temel arayuz sinifi
 */
class e_IsParcacigi : public e_Nesne
{
    public:

        e_IsParcacigi(SurecOnceligi_t oncelik);
        virtual ~e_IsParcacigi();

        /* tum is parcaciklari icin genel giris noktasi */
        static void *GenelGirisNoktasi(void *parametre);

        void basla();
        uint8_t onceligiDegistir(SurecOnceligi_t oncelik);

        /* tum is parcaciklari tarafindan gerceklesstirilecek temel metod */
        virtual void calismaNoktasi() = 0;

    protected:

        SurecKimligi_t kimlik;
        eGIS_SurecOzellikleri surecOzellikleri;
};
```

Şekil 8. eGİS uygulama iş parçacığı arayüzü

Şekil 8 gömülü uygulamalar için genel bir iş parçacığı arayüzünü göstermektedir. İş parçacıkları eGİS sisteminde *eGİS_Süreç* ile temsil edilirler. Ancak uygulamalar için daha genel, detayların saklandığı bir arayüz gerekmektedir. *e_IsParcacigi* işletim sistemi içsel detaylarını uygulamadan saklar. Bu tip arayüzler sayesinde uygulamalar işletim sistemi içerisinde ortaya çıkacak değişimlerden etkilenmezler. Ayrıca, hangi alt sistemin kayıt edilmiş olunduğu ve hangi donanımsal ortamda çalışıldığı bilinmeden, uygulamalar isteklerini karşılayabilirler.

```
/**
 * Ornek uygulama
 */
class Uygulama_1 : public e_IsParcacigi
{
    public:

        Uygulama_1();
        virtual ~Uygulama_1();

        virtual void calismaNoktasi();
};
```

Şekil 9. eGİS uygulama arayüzünden türeyen bir uygulama sınıfı

Şekil 9, e_IsParcacigi arayüzünü gerçekleştiren bir uygulama sınıfını göstermektedir. Her işparçacığı, saf sanal fonksiyon olan ***calismaNoktasi*** metodunu gerçeklemelidir. Görüldüğü gibi uygulama sınıfında da, bu metod gerçekleştirilmiştir. Bu metod içerisinde uygulamanın işleyeceği kodlar tanımlanmıştır.

```
#include <eGIS_programlamaarayuzu.h>
#include "uygulama_1.h"

using namespace eGIS;

e_Kilit kilit_1;
e_Kilit kilit_2;

/**
 * uygulamalarin baslayacagi nokta
 */
void *uygulamaBaslangici(void *arg)
{
    Uygulama_1 uygulama_1;

    cout<<"UYGULAMA BASLADI...\n";

    kilit_1.kilitle(0);
    cout<<"Kilit 1 kilitlendi...\n";

    kilit_2.kilitle(0);
    cout<<"Kilit 2 kilitlendi...\n";

    uygulama_1.basla();

    while(1)
    {
        cout<<"Ana Uygulama ...\n";
    };
}
```

Şekil 10. Uygulamaların başlatılması

Tanımlanan uygulamalar, uygulamalar için bir başlangıç noktası olan ***uygulamaBaslangici*** fonksiyonu içerisinde başlatılırlar. İşletim sistemi zaten çalışmaya başlamıştır ve yönetim sistemleri bu noktada aktiftir. Şekil 10'da gösterildiği gibi uygulamaların başlatılması ile, işletim sistemi hangi iş parçacığı en öncelikli ise işlemciyi ona verecektir. Bu iş parçacığı işlemciyi kendisi bırakana dek ya da daha öncelikli bir iş parçacığı yaratılana dek işletim sistemi bu iş parçacığını çalıştıracaktır.

Desen	*İşlem*	*Desen katmanında harcanan saat darbesi*	*Geçen toplam saat darbesi*
Tekil	eGIS_Mikroçekirdek nesnesine erişim	12	12
Köprü	Bir sonraki adımda çalışacak iş parçacığının belirlenmesi ve çalıştırılması	29	168
Önyüz	İş parçacığı yaratımı	< 234	1146

Tablo 1. eGİS'in i386 mimarisinde çalıştırılması sonucunda elde edilen deneysel sonuçlar

Tasarım desenlerinin uygulanması sistemde oluşan ek yük Tablo 1'de gösterilmiştir. Bu sonuçlar Bochs emülatörü üzerinde alınmıştır. Örneğin *eGİS_Mikroçekirdek* örneğine erişmek 12 saat darbesi sürmektedir. Sistemin çok duyarlı zaman kısıtına sahip gerçek zaman gereksinimlerine ihtiyaç duyan noktaları dışında, bu desen güvenli bir şekilde kullanılabilir. Süreç yönetimi köprü deseninin kullanıldığı bir diğer sistem noktasıdır. Bir sonraki adımda hangi sürecin çalışacağına karar verilmesi yaklaşık 168 saat darbesi sürmektedir. Süreçler arası geçiş işleminde köprü katmanından geçiş 29 saat darbesi sürmektedir. Eğer gerçekleştirimlerin ve arayüzlerin ayrılması ile ortaya çıkan avantajları ve özellikle de donanımsal ortamlara taşınabilmenin kolaylaşması göz önüne alınırsa, köprü deseninin getirmiş olduğu 29 saat darbelik ek yük kabul edilebilir boyutlardadır. Önyüz katmanı üzerinden bir iş parçacığı yaratımı isteğinin ilgili alt sisteme iletilerek işlenmesi yaklaşık 1146 saat darbesi tutmaktadır. İstek önce önyüz katmanında işlenmekte ve buradan süreç yönetim sistemine iletilmektedir. Ön yüz katmanında geçen süre 234 saat darbesidir. Burada önyüz katmanında yapılan işlemlerin hepsinin bu desene özgü işlemler olmadığı belirtilmelidir. Örneğin iş parçacığı yaratım parametrelerinin eGİS mikroçekirdeğine geçirilmesi bu desen olmasa da yapılmak zorundadır. Dolayısıyla, 234 saat darbelik sure, önyüz katmanı için harcanan sürenin üst sınırını oluşturmaktadır.

Bize göre Tablo 1 'deki başarım sonuçları, desenlerin getirdiği ek yüklerin özellikle işlemcilerin daha güçlendiği gömülü sistemlerde kabul edilebilir boyutlarda olduğunu göstermektedir.

6. Sonuç

Bu çalışmada, yeni bir gerçek zamanlı ve gömülü işletim sistemi çekirdeği olan eGİS, tasarım desenleri göz önüne alınarak gerçekleştirilmiştir. eGİS'in

uygulama gereksinimlerine göre değişebilir ve şekillendirilebilir bir sistem olacak şekilde tasarlanmasına dikkat edilmiştir.

Tasarım desenlerinin eGİS sistemi içerisinde uygulanmasının çözmüş olduğu problemler aşağıdaki gibi özetlenebilir:

- Nesneleri somut sınıf isimlerini açık bir şekilde belirterek yaratma *Tekil* ve *Soyut* Fabrika desenleri ile ortadan kaldırılmıştır.
- *Köprü* yapısal deseni nesneler arasındaki bağımlılıkları ve ortam bağımlılıklarını ortadan kaldırmıştır.
- *Önyüz* deseni eGİS işletim sisteminin içsel yapısını istemcilerden soyutlar ve temiz bir uygulama arayüzü sunar.
- *Strateji* deseni yönetim algoritmalarının birbirleri ile değiştirilebilmesini ve eGİS sisteminin yönetim algoritmalarından bağımsız olması sağlanmıştır.

Gelecekte yapılacak çalışmalarda eGİS'e yeni alt sistemlerin eklenmesini planlamaktayız. Bellek ve disk üzerinden çalışabilen bir dosya sistemi, TCP/IP yığıtı, gömülü kullanıcı arayüzü sistemi gibi alt sistemlerin eklenmesi eGİS'i daha da işlevsel hale getirecektir. Yeni yönetim algoritmalarının sisteme eklenmesi, eGİS'i daha yüksek oranda uygulama gereksinimlerini karşılayabilecek hale getirecektir. Ayrıca eGİS mimarisinin daha da iyileştirilmesi ve eniyilenmesi diğer hedeflerden biridir. Ek olarak eGİS'in farklı donanım ortamlarına da taşınması gerekmektedir.

eGİS işletim sisteminin, Türkiye'de açık kaynak koda sahip ve GNU lisansı ile diğer mühendislerin de katkılarına açık bir ulusal gömülü işletim sistemi olması umulmaktadır. Tıpkı eCOS işletim sistemi gibi, eGİS de diğer mühendislerin katılımı ile, daha hızlı ve sağlam bir genişleme imkanına sahip olacaktır. Bu amaçla eGİS açık kaynak kodlu projelerin yer aldığı www.sourceforge.net sitesinde paylaşıma açılmıştır [18].

Sonuç olarak yapılan çalışma ile, bu konu hakkında çalışma yapmak isteyenler için bir kaynak ortaya konulmuştur. Bu kaynağın, işletim sistemleri ve gömülü sistemler ile ilgilenenlere faydalı olacağı umulmaktadır.

7. Referanslar

[1] E.A. Lee, "What's Ahead for Embedded Software? ", *IEEE Computer Society Press, Computer, v.33 n.9 pp.18-26,* 2000.

[2] M. Gien, "Next Generation Operating Systems Architecture", *In Proceedings of the International Workshop on Operating Systems of the 90s and Beyond*, 1991.

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, 1994.

[4] B. P. Douglass, *Realt-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*, Addison-Wesley, 2002.

[5] J. Liedtke, "On micro-kernel construction", *In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[6] K. S. Yıldırım, "Gömülü Sistemler İçin Tasarım Desenleri Kullanarak Nesneye Yönelik, Gerçek Zamanlı Bir Mikroçekirdek Tasarımı", *Yüksek Lisans Tezi*, *Ege Üniversitesi Bilgisayar Mühendisliği Bölümü*, 2005.

[7] A. S. Tanenbaum, *Modern Operating Systems*, Prentice Hall, 2001.

[8] J. J. Labrosse, *MicroC/OS-II: The Real-Time Kernel, 2nd Ed.*, CMP Books, 2003.

[9] The Nucleus Plus RealTime Operating System anasayfası: www.acceleratedtechnology.com/embedded/nuc_rtos.html, 2001.

[10] The eCOS Operating System anasayfası: http://ecos.sourceware.org/

[11] U. Haack, W. Schröder-Preikschat, F. Schön, O. Spinczyk, "Design Rationale of the PURE Object-Oriented Embedded Operating System", *In Proceedings of The International IFIP WG 10.3/WG 10.5 Workshop on Distributed and Parallel Embedded Systems,* 1998.

[12] K. Levchenko, "A Survey of Microkernel Operating Systems", http://www.cs.ucsd.edu/classes/fa01/cse221/projects/index.html

[13] D. Gay, P. Levis, D. Culler, "Software Design Patterns for TinyOS", *In Proceedings of the LCTES05*, 2005.

[14] A.A. Fröhlich, W. Schröder-Preikschat, "EPOS: an Object-Oriented Operating System", *In Proceedings of the Workshop on Object-Oriented Technology, Lecture Notes In Computer Science Volume 1743*, 1999.

[15] The GNU Compiler Collection anasayfası: http://gcc.gnu.org/

[16] The GNU Project Debugger anasayfası: http://www.gnu.org/software/gdb/

[17] Bochs IA32 PC Emulator homepage anasayfası: http://bochs.sourceforge.net/

[18] eGİS İşletim Sistemi kod ambarı adresi: http://sourceforge.net/projects/egis-eeos/

Dağıtık Sistemler için Eşden Eşe (Peer-to-Peer) Mimarisine Dayalı Hata Yakalama Modeli

Celal Kavuklu
ASELSAN A.Ş.
ckavuklu@mst.aselsan.com.tr

Öz

Dağıtık sistemlerin sağladığı servislerin hayatımızdaki önemi gün geçtikçe artmaktadır. Bu servislere örnek olarak birçok yazılım ve donanımın bir araya gelerek sağladığı internet servisleri (arama motorları, elektronik bankacılık), telekom sistemleri (internet bağlantı servisleri), komuta kontrol uygulamaları, vb verilebilir. Sağlanan servislerin sürekli işlevsel tutulabilmesi için gerekli ilk adımın servisi sağlayan bileşenlerde oluşacak hataların yakalanması olduğu değerlendirilmektedir. İlgili servislerin çeşitli sayılarda olması, çeşitli ölçeklerde çalışması ve değişik hata tanımlarına sahip olması, dağıtık sistemler için genelleştirilmiş bir hata yakalama yöntemi geliştirilmesini zorlaştırmaktadır. Bildiride, dağıtık sistemlerde sunulan servislerin hatalarının yakalanabilmesi için oluşturulan, eşden eşe mimarisine dayalı, genelleştirilmiş bir hata yakalama modeli sunulmuş ve bu modelin uygulanabilirliği anlatılmıştır.

1 Giriş

Bir dağıtık sistem servisi, çeşitli sayıda donanım ve yazılım bileşeninin beraber çalışması ile oluşturulur. Servis, sunduğu işlevselliği kullanacak kullanıcılar (başka servisler olabilir) için, bir servis ulaşım arayüzü sunar. Servislerin hatalı olup olmadıkları, servis ulaşım noktalarında sergiledikleri davranışların beklenildiği gibi olup olmaması ile değerlendirilir. Servislerin, ulaşım noktalarında sergiledikleri sapmalar, ilgili noktaların sürekli izlenmesiyle ve/veya servisi oluşturan bileşenlerin durumlarının sürekli incelenip analiz edilmesiyle çıkarılabilir. Örneğin, web sunucusu ve veri tabanı bileşenlerinden oluşan bir web sistemini ele alalım. Sistemin sunduğu bir web servisinde hata oluşup oluşmadığı, bir tarayıcı yardımıyla servis ulaşım arayüzü olan IP ve port numarasına erişilerek görülebilir. Bu yöntemle, hata oluştuktan ve servis arayüzüne yansıdıktan sonra hata fark edilmektedir. Bunun yanında, ilgili web servisini oluşturan veri tabanı ve web sunucusu bileşenlerinin durumlarının izlenmesi ile; serviste oluşacak hatalar, servis arayüzüne yansımadan tahmin edilebilir.

Bu bildiride sunulan çalışmada, servis hatalarının, ilgili servisleri oluşturan bileşenlerin durumları analiz edilerek bulunması yöntemi uygulanmıştır. Bu çalışmadaki temel amaç, dağıtık sistemlerin sunduğu farklı özelliklerdeki (ağ yapısı, ölçek, haberleşme protokolü, performans gereksinimi, vb) servislerin hata durumlarının, servisi oluşturan bileşenlerin analizi ile bulunduğu genelleştirilmiş bir hata yakalama modeli sunmaktır.

Dağıtık sistemler için sunulan genelleştirilmiş bir hata yakalama modelinin, çeşitli özelliklere sahip olması; başka bir deyişle, çeşitli gereksinimleri karşılaması gerekmektedir. Sunulan model bu gereksinimler göz önünde bulundurularak hazırlanmıştır. İlgili gereksinimleri karşılayacak işlevsellik, eşden eşe mimarisinin getirdiği kazanımlar kullanılarak elde edilmektedir.

Eşden eşe mimarisi, dağıtık sistemlerin karşılaştığı çeşitli kısıtların (ölçeklenebilirlik ve fiziksel ağlara bağımlılık) etkilerini en aza indirmek için tasarlanmıştır. Mimarinin altyapısını, varolan bir fiziksel ağ üzerine kurulu kendi kendine organize olabilen katmanlı bir ağ (self-organizing overlay network) oluşturmaktadır. Bu ağ katmanı üzerinde çalışan protokoller, eşler (peer) ve eş grupları (peer group) mimariyi tamamlamaktadır. Eş grupları, aynı işle ilgilenen eşlerin, katmanlı ağ üzerinde oluşturduğu sanal bir buluşma katmanı olarak düşünülebilir. Eş grupları, kendi gruplarına üye olan eşlere, çeşitli servisler (peer group service) sunabilirler.

Sunulan modelde, dağıtık sistem servislerini oluşturan bileşenler, eşden eşe mimarisinde birer eş olarak soyutlanmaktadır. Aynı dağıtık sistem servisi için ortaklaşa çalışan eşler, bir araya gelerek, içinde

ilgili servis için hata yakalama mekanizmalarını barındıran eş gruplarını oluşturmaktadır.

2 Hata Yakalama Gereksinimleri

Hata yakalama modeli oluşturulurken göz önünde bulundurulan gereksinimler, daha önce yapılan benzer çalışmalardan derlenmiştir [1][2].

2.1 Ölçeklenebilirlik

Dağıtık sistemler, sistemde sağlanan servisleri oluşturan bileşenlerden oluşmaktadır. Bazı sistemler pek çok sayıda servis ve bileşen içerebilmektedir. Dağıtık sistemlerde oluşacak hataları yakalamak için sunulan bir yöntem, sistemdeki servis veya bileşen sayısından bağımsız bir şekilde işlevini gerçekleştirebilmelidir. Ölçeklenemeyen bir hata yakalama modelinin, büyük ölçekli sistemler için bir çözüm sunamayacağı açıktır.

2.2 Uyumluluk

Dağıtık sistemler, genel olarak asenkron ağlar üzerinde (mesajların ulaşma zamanı/gecikme zamanı hakkında tahmin yapılamayan ağlar) çalışmaktadır. Bu tip ağlarda çalışan sistemler için sunulan bir hata yakalama modeli, ağlardaki değişik durumlara (yoğun trafiğe bağlı mesaj gecikmeleri, değişik ağ topolojileri, ağlardaki dinamizm, haberleşme yöntemleri) kendini uyumlandırabilmelidir.

2.3 Esneklik

Hata yakalama modeli, değişik tipte hata yakalama gereksinimleri olan servisler için çeşitli hata yakalama mekanizmalarını (hata yakalama yöntemi, hata iletim yöntemi, vb) sağlayabilmelidir. Örneğin, dağıtık sistem servislerinden birisi, üyelik temelli bir hata yakalama mekanizmasını tercih ederken, bir diğeri güvenilmeyen hata yakalama yöntemini tercih edebilmelidir.

2.4 Servis Kalitesi

Hata yakalama modeli, sunduğu hata yakalama servisi kalitesini istenilen değerlerde tutacak mekanizmalar içermelidir. Hata yakalama alanındaki temel servis kalitesi parametreleri : hata yakalama hızı, hata yapma oranı (accuracy for failure detection), hata sorgusunun doğru olma ihtimali (query accuracy probability) şeklinde sıralanabilir. Örneğin: bir dağıtık sistem servisi için yüksek hızlarda hata yakalama yapması istenen bir hata yakalama yöntemi, değişen ağ trafiğine göre ilgili bileşenlerini sorgulama hızını değiştirebilmelidir.

2.5 Güvenlik

Güvenlik ihtiyaçları ön planda olan dağıtık sistemler için sunulan hata yakalama yöntemleri, sistemlerin güvenlik gereksinimlerini karşılamalıdır. Güvenlik ihtiyaçları özellikle geniş ağlar üzerinde çalışan sistem ve servisler için ön plandadır [1]. Örneğin: hata yakalama modeli, geniş ağda çalışan bir komuta kontrol servisinin bileşenlerinin durumlarını, ancak gerekli kimlik denetimi (authentication) ve yetkilendirme (authorization) kurallarını yaptıktan sonra değerlendirmelidir.

2.6 Bağımlılık Analizi

Bir hata yakalama yöntemi, sistemdeki servislerin durumunu değerlendirirken; sistemdeki servislerin, servisi oluşturan bileşen ve alt-servislere olan bağımlılıklarını otomatik bir şekilde değerlendirebilmelidir. Örneğin: dağıtık sistemdeki A servisi doğru çalışmak için B servisine ve C bileşenine bağımlıysa; ilgili hata yakalama yöntemi, B servisinde veya C bileşeninde bir hata yakaladığında, A servisini de hatalı ilan etmelidir.

3 Benzer Çalışmalar

Bu başlık altında, hata yakalama alanında yapılan benzer çalışmalar, listelenen gereksinimler göz önünde bulundurularak incelenmektedir.

3.1 FUSE

FUSE [1], dağıtık sistemler için oluşturulan bir hata bildirme sistemidir ve kendi kendine organize olabilen katmanlı bir ağ üzerine (SkipNet [3]) kurulu etkili bir haberleşme katmanından oluşmaktadır. Kullanılan katmanlı ağ ve haberleşme katmanı: ölçeklenebilirlik, uyumluluk, servis kalitesi ve bağımlılık analizi gibi konularda gerekli işlevselliği sağlamaktadır. Buna karşılık, esneklik ve güvenlik konularında yeterli çözümler üretememektedir.

FUSE sistemi üzerinde, değişik tipte hata yakalama mekanizmalarının gerçeklenebileceği altyapı bulunmaktadır. Bunun yanında, bir dağıtık sistem içinde sadece bir tipte hata yakalama mekanizması çalıştırılabilmektedir. Bunun sonucunda, hata yakalama mekanizmasına ait servis kalitesi ve hata iletim yöntemi, bir dağıtık sistem içinde sabit kalmakta, sistemdeki servislere göre esneklik gösterememektedir.

FUSE sisteminde, güvenliği arttırıcı alternatif yöntemler sunulsa da, kimlik kontrolü ve yetki analizi gibi konularda çözümler sunulmamaktadır.

3.2 Hata Bulma Servisi

Hata bulma servisi [2], hiyerarşik bir haberleşme altyapısı üzerinde, sırasıyla servis kalitesini ve güvenilmez hata yakalayıcı mekanizmasını kontrol eden katmanlardan oluşmaktadır. Servisin, hiyerarşik haberleşme altyapısı üzerinde kullandığı katmanlı yapı: ölçeklenebilirliği, bağımlılıkları etkili bir şekilde çözmeyi ve gerekli servis kalitesini sağlamaya yöneliktir. Bunlarla birlikte, sistemin fiziksel ağa bağımlılığını azaltan haberleşme protokolleri (Jini [24]) kullanması, sistemin uyumluluğunu arttırmaktadır.

Hata bulma servisi, üyelik temelli hata yakalama mekanizmasını desteklemediği için esneklik gereksinimini yerine getirememektedir. Buna ek olarak, ilgili sistemde güvenlik konusunda herhangi bir çözüm sunulmamaktadır.

4 Model Mimarisi

Sunulan hata yakalama modelinin temelinde, dağıtık sistemdeki her servis için bir hata yakalama servisinin bir eş grubu servisi şeklinde sunulması yer almaktadır. Hata denetimi yapılan servisleri oluşturan her bileşen ve alt-sistem, bu eş grubunun içinde birer eş şeklinde temsil edilmektedir. Hata yakalama servisi, çalıştığı eş grubu içindeki eşlerin durumlarını çeşitli hata yakalama mekanizmaları kullanarak tespit eder ve bu durumlardan hata yakalama hizmeti sunduğu dağıtık sistem servisinin durumunu oluşturur.

Model oluşturulurken ilham alınan çalışma [5] da, sistem servislerinin hatalarının yakalanabilmesi için eş grubu içinde tanımlanan hata yakalama servislerini kullanmıştır. Buna karşın, ilgili çalışmadaki hata yakalama servisleri, sistem servislerinin hatalarını, servislerin ulaşım arayüz noktalarını izleyerek yakalamayı hedeflemektedir.

Hata yakalama modeli mimarisi, eş grubu servislerinin ölçeklenebilirliğini arttıran [4] mimarisini temel almaktadır.

4.1 Eşler

Hata yakalama modelinde, üç farklı tipte eş yer almaktadır: işçi eş, istemci eş ve randevu eş.

İşçi Eş: İşçi eşler, kendi iş grupları içinde, hata yakalama modeli içindeki hata yakalama mekanizmalarını gerçekleştirirler. Mimaride, bir iş grubu içinde birden fazla işçi eş bulundurularak hata yakalama mekanizmalarının sürekli işlevselliği sağlanmıştır.

İstemci Eş: İstemci eşler, sistemdeki servisleri oluşturan bileşenler ile hata yakalama servisi arasında köprü görevi yapmaktadırlar. İstemci eşler, bileşenlerin durumlarını, bileşenin sağladığı sistem servisine ait eş grubuna üye olarak ilgili eş grubundaki hata yakalama servisine aktarırlar.

Randevu Eş: Randevu eşler, modeldeki hata yakalama servislerinin / eş gruplarının bulunması ve sistemdeki eşlerin durumlarının belirlenmesi için kullanılmaktadır.

4.2 Eş Grupları

Modeli oluşturan eşler, bir araya gelerek eş grupları oluştururlar. Farklı eş gruplarının kullanılmasının temel amacı, sunulan farklı işlevler için farklı kapsamlar oluşturmaktır.

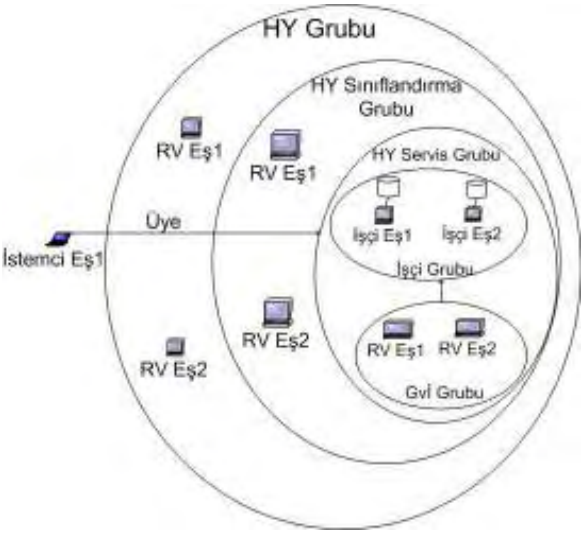
İşçi Grubu: İşçi grubu, servis grubu içindeki işçi eşleri gruplamak için kullanılmaktadır. Bir işçi grubundaki tüm işçi eşler hata yakalama servis grubunda sunulan hata yakalama mekanizmasını gerçeklemekte ve işçi grubu içindeki haberleşme servisleri ile haberleşmektedir.

Giriş ve İzleme (Gvİ) Grubu: Bu grup, randevu eşleri gruplamak için kullanılmaktadır. Gvİ grubu, hata yakalama modelindeki işçi eşlere iş paylaştırma ve işçi eşlerin izlenmesi gibi işlevleri yerine getirmektedir. Bu işlevler ile, modelin ölçeklenebilirlik özelliği arttırılmaktadır.

Servis Grubu: Servis grubu, modelin uygulandığı dağıtık sistemdeki her servis için bir adet bulunur. Servis grupları, bünyelerinde, o servise uygulanacak hata yakalama servisini çalıştıran işçi grubunu ve işçi grubuna erişimi sağlayan ve durumunu değerlendiren Gvİ grubunu barındırırlar.

Sınıflandırma Grubu: Bu grup, hata yakalama servisi gruplarının arama kapsamını daraltmak için kullanılır. Arama kapsamının daraltılması, istenen grup veya servislerin daha hızlı konumlandırılmasını sağlamaktadır.

Modeli oluşturan eşler ve eş grupları, bir sonraki sayfada yer alan şekille (Şekil-1) görselleştirilmiştir.



Şekil 1 Eş / Eş grubu mimarisi

4.3 Servisler

Model, temel eşden eşe servislerinin kullanımına dayalıdır. Bu servislerin protokolleri gerçeklemeden gerçeklemeye değişmesine rağmen genel mantık ve isimlendirme hepsinde aynıdır. Örneğin:

- İletişim, keşif ve konumlandırma servisleri, bir belirteci (identifier) eşden eşe ağındaki bir düğüme eşlemekte ve bu düğüme bir içerik göndermede kullanılır.
- Üyelik servisleri, eşlerin eş gruplarına üyelik işlemlerini gerçekleştirmek için kullanıldığı gibi üyelik tipi hata yakalama mekanizmaları için de altyapı oluşturur.
- Güvenlik servisleri, eşlerin eş grubuna üyeliği sırasındaki kimlik kontrolü, yetkilendirme gibi işlevlerin yanı sıra, güvenli haberleşme fonksiyonlarını sağlamak için de kullanılır.

Temel eşden eşe servisleri, modeldeki tüm eşlerde çalışmaktadır. İlgili servislerin yönetimi, altta bulunan eşden eşe sistemine yapılan çağrılar ile yapılmaktadır.

5 Modelin Analizi

Hata yakalama modelinin kullandığı eşden eşe protokolleri, modele aşağıdaki yetenekleri kazandırmaktadır:

- İletişim, keşif ve konumlandırma servislerinin kullanımı, oluşturduğu katmanlı ağ ile uyumluluk, servis kalitesi ve ölçeklenebilirlik açısından modele avantajlar kazandırmaktadır. Katmanlı ağ, modelin fiziksel ağa bağımlılığını ortadan kaldırmakta, farklı eş grupları için düşük ek yüklü farklı katmanlar yaratarak, servis kalitesini ve ölçeklenebilirliği arttırmaktadır.
- Üyelik servislerinin kullanımı, bağımlılık analizinde modelin ölçeklenebilirliğini arttırmaktadır. Bunun nedeni, eşden eşe protokollerinin ölçeklenebilir olması ve düşük bir ek yük oluşturmasıdır. Buna karşılık, üyelik servisleri ölçeklenebilirliği arttırmak amacıyla genel olarak üyeleri gevşek bir bağ ile (loosely coupled) birbirine bağlamaktadır. Bu haliyle, üyelik servisleri, tutarsız üyelik tipi hata yakalama sistemleri (weakly consistent membership) için altyapı oluşturmaktadır [5][6][7]. Üyelik servislerindeki bağ, üyelik tazeleyici mekanizmalarla geliştirilirse (ölçeklenebilirlikten feragat edilerek) , tutarlı üyelik tipi hata yakalama mekanizmaları (strongly consistent membership) için altyapı oluşturmaktadır.
- Temel eşden eşe servisleri üzerine çeşitli güvenlik protokolleri [8][9][10] uygulanabilmektedir. Modelde kullanılan eş grupları, ilgili güvenlik çözümlerinin her servis genelinde uygulanabilmesi için uygun birer kapsam oluşturmaktadır. Bu durum, modeli benzer çalışmaların aksine, kullanıldığı sistemlerdeki güvenlik ihtiyaçlarını karşılayacak seviyeye getirmektedir.

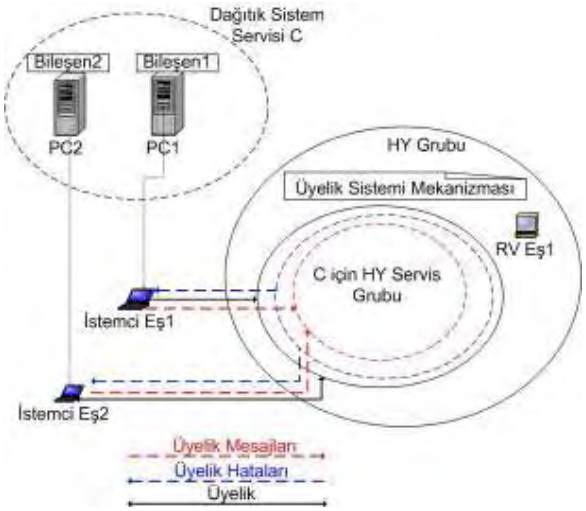
Hata yakalama modelinde, her dağıtık sistem servisi için bir hata yakalama servis grubu bulunması; modele, her servise değişik tipte hata yakalama hizmeti sunulabilmesi esnekliğini ve altyapısını kazandırmıştır. Örneğin, farklı hata yakalama hizmetleri olarak aşağıdaki maddeler listelenebilir:

- Hata yakalama mekanizmaları
- Hata iletim yöntemleri
- Güvenlik ihtiyaçları
- Servis kalitesi ihtiyaçları

Servis grubu içindeki işçi grubunda, işçi eşler tarafından gerçekleştirilecek değişik tipteki hata iletim yöntemleri (yayım / dedikodu temelli hata iletimi, üyelik görüntüleri aktarımı); modele, farklı servis kalitesinde çalışma altyapısını kazandırmaktadır. İstenen hata yakalama hızını yakalamak için yerel ağ sistemlerinde yayın (broadcasting) temelli iletim yöntemleri tercih edilirken; çok büyük ölçeklerde ve geniş ağ sistemlerinde çalışan servisler için anlık sorgu doğruluğundan feragat edilerek ölçeklenebilirliği çok yüksek olan dedikodu (gossip) temelli [11][12] bir hata iletim yöntemi tercih edilebilir.

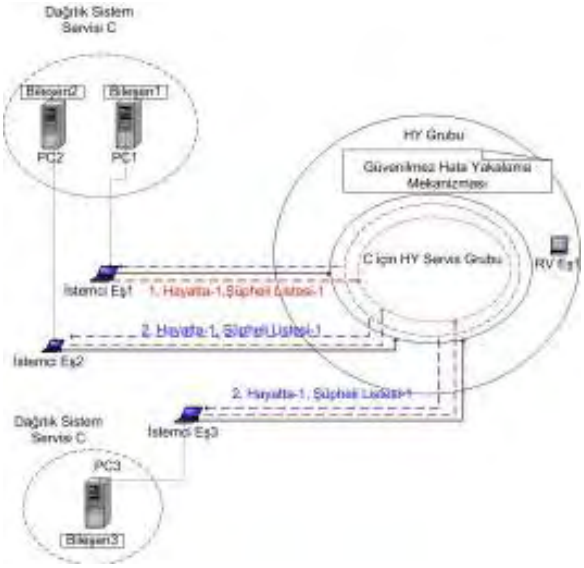
İşçi eşler tarafından gerçekleştirilecek değişik tipteki hata yakalama mekanizmaları (üyelik sistemleri, güvenilmez hata yakalayıcılar), benzer çalışmalardan farklı olarak, modele değişik dağıtık sistemlere uyum esnekliğini kazandırmaktadır. Model ile gerçekleştirilebilecek bazı hata yakalama mekanizmaları aşağıda açıklanmıştır:

Üyelik sistemi: Üyelik sistemi hata yakalama mekanizması ile, servisleri oluşturan bileşenler, modeldeki istemci eşlere eşlenir. İstemci eşler, ilgili servis için oluşturulan servis grubuna üye olur ve bileşenin durumunu üyelik mesajları ile servis grubuna aktarır. Servis grubunda kullanılan hata yakalama mekanizması ile, üyelik durumları kullanılarak serviste hata olup olmadığı analiz edilir ve varsa üyelik hata mesajı olarak istemci eşlere aktarılır. Şekil-2'de, üyelik sistemi mekanizmasının temel bir şekilde model tarafından nasıl gerçeklenebildiği görülmektedir.



Şekil 2 Üyelik Sistemi Hata Yakalama Mekanizması

Güvenilmez Hata Yakalayıcı: Güvenilmez hata yakalayıcı yönteminde, servisleri oluşturan bileşenlere eşlenen istemci eşler, eşlendikleri bileşenin durumlarını ve servisi oluşturan diğer bileşenlerle ilgili şüphelerini servis grubunda sunulan hata yakalama mekanizması ile diğer istemci eşlere aktarırlar. Bilgileri alan istemci eşler, kendi şüpheli listelerini oluştururlar. Güvenilmez hata yakalayıcı mekanizmasının en basit şekilde nasıl model tarafından gerçeklenebildiği Şekil 3'de gösterilmektedir. Resmedilen yöntem incelenirken, ilgili resmin örnekleme amaçlı olduğu ve model tarafından gerçeklenebilecek tek güvenilmez hata yakalayıcı yöntemi olmadığı göz önünde bulundurulmalıdır. Gösterimdeki mesajların başındaki numaralar mesaj sırası, X-n şeklindeki mesaj gösterimleri ise, n numaralı bileşene ait X bilgisi olarak yorumlanmalıdır.



Şekil 3 Güvenilmez Hata Yakalama Mekanizması

6 Modelin Gerçeklenmesi

Hata yakalama modelinin gerçeklenebilmesi için birbirinin alternatifi veya bütünleyicisi olan birçok yöntem uygulanabilir. Alternatif yöntemler arasındaki seçim, modelin uygulanacağı dağıtık sistemin özelliklerine göre seçilir. Alt başlıklar altında, modele ait bölümlerin alternatif gerçeklenme yöntemleri tartışılmaktadır.

6.1 Bileşenlerin Eşe dönüşümü

Hata yakalama modelinde yer alan eşler, dağıtık sistemi oluşturan yazılım ve donanım bileşenlerinin eşden eşe ortamında soyutlanmış halleridir. Bu soyutlama için, bileşenlerin eşlere dönüşümü yapılmalıdır. İlgili dönüşümü gerçekleştirecek mantığın paketlenmesi için çeşitli yöntemler uygulanabilir. Uygulanabilecek paketleme yöntemlerinden birkaçı aşağıda sıralanmıştır:

- Kütüphane
- Katmanlı bir ara yazılım (middleware)
- Yazılım bileşeni

Dağıtık sistemin özelliklerine uygun bir şekilde seçilen paketleme mantığı, sistem içinde tek bir şekilde veya harmanlanarak gerçeklenebilir. Örneğin, sistemin

kısıtlarına göre aynı sistem içinde hem kütüphane hem de yazılım bileşeni kullanılarak dönüşüm sağlanabilir.

Yazılım Bileşenlerinin Eşe Dönüşümü: Dönüşüm mantığını bünyesinde barındıran bir kullanıcı kütüphanesi veya bir ara katman yazılımı, programlama arayüzleri yardımıyla, ilgili yazılım bileşenini (uygulama, işlem, iş parçacığı, vb) eşden eşe ortamında bir eşe dönüştürebilir. Bunun yanında, dönüşümü gerçekleştirecek bir başka yazılım bileşeni ise, iki bileşen arasında işlemlerarası bir haberleşme (Inter-process communication) veya ethernet/seri kanal arayüzleri kullanılarak dönüştürme işlemi gerçekleştirilebilir. Aşağıdaki şekilde, yazılım bileşenlerinin programlama arayüzleri yardımıyla eşe dönüşümleri görselleştirilmiştir.

Şekil 4 Yazılım Bileşeni – Eş Dönüşümü

Donanım Bileşenlerinin İstemci Eşe Dönüşümü: Model genel olarak yazılım bileşenleri göz önünde bulundurularak tasarlanmıştır. Bunun yanı sıra, yazılım bileşenlerinin dönüşümünü sağlayan mantığa, donanım bileşenlerinin, çeşitli sürücü veya işletim sistemi seviyesi fonksiyonları ile izlenmesi mantığı da eklenerek model genişletilebilir. Aşağıdaki şekilde, donanım bileşenlerinin izlenmesi mantığı kullanılarak, donanım bileşenlerinin istemci eşlere dönüşümü görselleştirilmiştir.

Şekil 5 Donanım Bileşeni – İstemci Eş Dönüşümü

6.2 Eşden Eşe Platformu

Sunulan model, temel eşden eşe protokollerini kullanmaktadır. Bu nedenle, model gerçeklenirken, bu protokollerin tanımlandığı bir çatı (framework) üzerine kurulmalıdır. Böyle bir çatı: kendi kendine organize olabilen ve eşden eşe temelli olan varolan katmanlı ağlar üzerine etkin altyapılar kurularak veya kapsamlı eşden eşe çatıları kullanılarak elde edilebilir.

Varolan eşden eşe temelli ağlar üzerine kurulan etkin yapılar, kullanılmakta olan eşden eşe katmanlı ağlar üzerine [13][14][15][3], etkin sorgulama [16], haberleşme [17][18][21] ve güvenlik [19][20] katmanları eklenerek elde edilebilir. Örneğin: dağıtık bilgi yönetim sistemi [21]; eşden eşe temelli ağ katmanı üzerine: veri yönetimi, güvenlik ve konu temelli bilgi üyeliği (topic based publish/subscribe) altyapıları geliştirmiştir.

Kapsamlı eşden eşe çatıları, tüm temel eşden eşe protokollerinin gerçeklendiği çatılardır. JXTA projesi [22], yaygın ve kapsamlı bir eşden eşe çatısı oluşturmaktadır. JXTA, katmanlı bir ağ yapısı sunmasının yanında:

- Eşden eşe protokollerini standardize eder ve sağlar,
- Eşden eşe ağının yönetilmesi ihtiyacını karşılayacak işlevselliği sunar,
- Platform bağımsızlığı sağlar. C ve JAVA desteği ile programlama dilinden bağımsızdır. Microsoft Windows ve UNIX türevi işletim sistemlerinde koşabilmektedir ve ağ protokollerinden (TCP/IP, Bluetooth) bağımsız çalışmaktadır [23].

7 Örnek Gerçekleme

Modelin gerçeklenmesinin anlaşılması ve işlevselliğinin ispatı amacıyla yapılan örnek bir gerçekleme çalışması, bu başlık altında anlatılmıştır. Yapılan gerçeklemede, JXTA çatısı kullanılmış ve modelin çalışması için gerekli mantık, bir kullanıcı seviyesi kütüphanesi içinde gerçeklenmiştir. Kütüphane en temel anlamda: servisleri oluşturan bileşen durumlarını eşden eşe ortamında sağlanan hata yakalama servislerine; servis sonucu bulunan hataları da, ilgili bileşenlere aktarma işlevlerini sağlamaktadır. Geliştirilen kütüphane, modeldeki tüm eşler için ayrı arayüzler ve işlevler sunmaktadır.

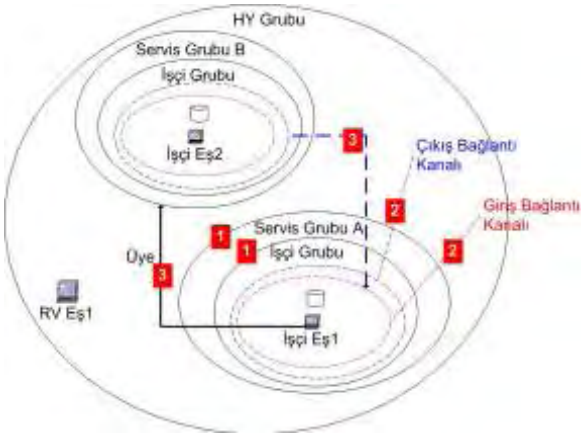
7.1 İşçi Eş İşlevleri

Kütüphane, sağladığı işçi eş işlevleri ile, yazılım bileşenlerini modeldeki işçi eşlere dönüştürür. Yazılım bileşenleri, hataları yakalanacak dağıtık sisteme ait bir bileşen veya hata yakalama için tasarlanıp yerleştirilmiş ayrı bir bileşen olabilir. Bu bileşenlere kütüphane tarafından sunulan işlevler: sınıflandırma grubu yaratma, servis grubu yaratma, gruplara üye olma, hata yakalama mekanizmalarının altyapılarını yaratma/yönetme, vb. Oluşturulan kütüphanede, bir işçi

eşin hata yakalama mekanizması altyapısını yaratma işlemi aşağıdaki adımlardan oluşmaktadır:

1. **Gruplara Üye Ol**: Keşif servisi kullanılarak ilgili servis grubunun ve işçi grubunun var olup olmadığı sorgulanır ve eğer varsa gruplara üyelik işlemleri yapılır. Gruplar bulunamazsa, her grup için JXTA grup reklamları (group advertisement) hazırlanarak keşif servisine yayınlanır ve üyelik işlemleri başlatılır.
2. **Hata Yakalama Servisi Yarat**: İlgili işçi grubunun içinde, hata yakalama mekanizmasının bir eş grup servisi şeklinde tanımlanması için: modül sınıf/belirtim reklamlarının (module class/spec advertisements) ve giriş/çıkış bağlantı kanalları (propagate communication pipes) hazırlanarak keşif servisine yayınlanır. Bağlantı kanalları bir belirteç ile tanımlanan yapılardır. Bu belirteçlerin, bilinen ağ protokollerine (TCP/IP) ve haberleşme parametrelerine (IP, Port) çevrimi JXTA protokolleri tarafından gerçekleştirilmektedir. İlgili bağlantı kanallarından: giriş bağlantı kanalı üzerinden bağımlı olunan alt servis veya bileşenlerle ilgili durumlar alınırken, çıkış bağlantı kanalına analiz edilen veriler sonucu oluşan servis durumu bilgileri gönderilir.
3. **Bağımlı Olunan Gruplara Üye Ol**: İşçi eşler, kendi servisleri için hata yakalama servis gruplarına üye olurken, aynı zamanda bağımlı olunan servisler için eşden eşe ortamında yaratılan hata yakalama gruplarına da üye olmalı ve bu gruplardaki hata yakalama mekanizmalarını kullanarak bağımlı servislerin durumunu öğrenmelidir. Bunun için bağımlı olunan servislere ait servis gruplarına üyelik işlemi yapıldıktan sonra, ilgili gruptaki hata yakalama mekanizmasının çıkış bağlantı kanalına bağlanılarak servis durumları elde edilir.

İlgili hata yakalama altyapısı kurulduktan sonra, işçi eşler, gerçekledikleri hata yakalama mekanizmalarını ilgili altyapı üzerinde kullanabilir hale gelmektedirler. Aşağıdaki resimde, B servisine bağımlı A servisi için ilk defa hata yakalama mekanizması altyapısını yaratma işleminin sonucu resmedilmektedir. Nesneler üzerindeki numaralar, nesnelerin yaratılmalarının hangi adımlarda gerçekleştiğini açıklamaktadır. Örneğin, A servisi için servis ve işçi grupları 1.adımda yaratılmıştır.



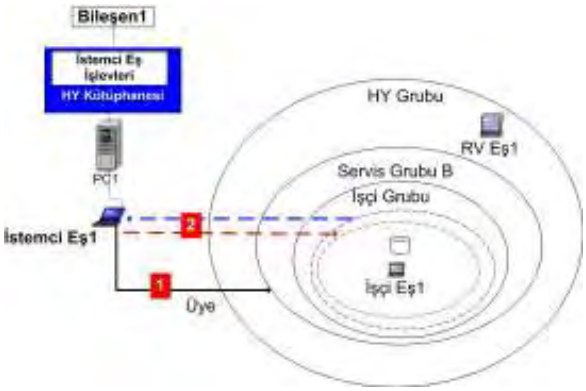
Şekil 6 Örnek Hata Yakalama Mekanizması Yaratımı

7.2 İstemci Eş İşlevleri

Kütüphane, sağladığı istemci eş işlevleri ile, dağıtık sistemdeki yazılım bileşenlerini modeldeki istemci eşlere dönüştürür. Bu bileşenlere kütüphane tarafından sunulan işlevler: gruplara üye olma, hata yakalama mekanizmalarını kullanma, vb. Oluşturulan kütüphanede, bir istemci eşin, hata yakalama mekanizması kullanma işlemi aşağıdaki adımlardan oluşmaktadır:

1. **Gruplara Üye Ol**: Keşif servisi kullanılarak ilgili servis grubunun ve işçi grubunun var olup olmadığı sorgulanır ve eğer varsa gruplara üyelik işlemleri yapılır. Gruplar bulunamazsa periyodik sorgularla aranmaya devam edilir.
2. **Hata Yakalama Servisini Kullan**: Hata yakalama servisinin kullanılması için, ilgili servis grubuna üye olunduktan sonra hata yakalama servisinin bağlantı kanallarına bağlanılması gerekmektedir. Bu amaçla bağlantı kanallarının reklamları sorgulanır ve gelen cevaplar anlamlandırılarak giriş ve çıkış bağlantı kanallarına bağlanılır. İlgili bağlantı kanalları bulunamazsa, periyodik sorgularla aranmaya devam edilir. Bu aşamadan sonra, programlama arayüzü yardımıyla elde edilen bileşene ait durum bilgileri giriş kanalı üzerinden gönderilirken, çıkış kanalı üzerinden servisin genel durumu ile ilgili bilgiler alınır.

Bir sonraki sayfada yer alan şekilde (Şekil-7), B servisini sağlamak için işlev gösteren Bileşen-1'in ilgili servise ait hata yakalama mekanizmasını kullanması resmedilmektedir.



Şekil 7 Örnek Hata Yakalama Mekanizması Kullanımı

7.3 Randevu Eş İşlevleri

Kütüphane, randevu eş işlevleri için ek olarak herhangi bir yöntem sunmamaktadır. Kütüphaneyi yükleyen herhangi bir bileşen, kütüphaneye ait işçi ve istemci eş işlevlerinden birini kullanmazsa, randevu eşi olarak görev yapmaktadır. Kütüphane, modelin ispatı amacıyla geliştirildiği için, ölçeklenebilirliği arttırıcı GvĬ grubu gerçeklenmemiş; bunun yerine herhangi bir gruba veya servise bağlı olmayan genel randevu eşleri kullanılmıştır. Randevu işlevi, JXTA çatısı altında da aynı isimle bulunan randevu eşlerinin (Rendezvous Peer) sunduğu randevu servisi ile gerçekleştirilmektedir.

8 Sonuç

Bu bildiride, dağıtık sistemlerdeki servis hatalarının bulunabilmesi için eşden eşe temelli bir hata yakalama modeli sunulmuştur. Model ve bu alanda varolan benzer çalışmalar, dağıtık sistemler için sunulan bir hata yakalama modelinin sağlaması gereken özellikler çerçevesinde analiz edilmiştir. Yapılan analizlerde, modelin teorik olarak gerekli altyapıyı sağladığı değerlendirilmiştir.

Bildiride, sunulan modelin nasıl gerçeklenebileceğine dair açıklamalara yer verilmiştir. Bu açıklamalara ek olarak, modelin JXTA eşden eşe çatısı kullanılarak yapılan örnek gerçeklemesinin pratik ayrıntıları verilmiştir. Örnek gerçekleme kapsamında geliştirilen kütüphanenin, belirtilen kısıtlara sahip olsa da, değişik tipteki hata yakalama mekanizmalarını güvenli ve esnek bir şekilde uygulayabilecek altyapıyı oluşturduğu, sistemdeki bağımlılık analizlerini başarı ile yaptığı gözlenmiştir. Bununla beraber, kütüphane, teorik olarak uyumluluk, ölçeklenebilirlik ve servis kalitesi konusunda gerekli altyapıyı içermesine rağmen pratik anlamda bu konulardaki yeterliliğinin denenmesi sonraki çalışmalara bırakılmıştır. Bu bağlamda, yapılan örnekleme çalışması, bu alanda daha sonra yapılabilecek çalışmalar için önemli bir kaynak oluşturmaktadır.

9 Referanslar

[1] John Dunagan, Nicholas J. A. Harvey, Marvin Theimer, Michael B. Jones, Alec Wolman, and Dejan Kostic. “FUSE: Lightweight Guaranteed Distributed Failure Notification”. In Proc. of OSDI 2004, December 2004.

[2] Bruno G. Catão , Francisco V. Brasileiro ,and Ana Cristina A. Oliveira. “Engineering a Failure Detection Service for Widely Distributed Systems”. In Proc. of SBRC’05, Brazil, 2005.

[3] Nicholas J.A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. “SkipNet: A Scalable Overlay Network with Practical Locality Properties”. In Proc. of 4th USENIX Symposium on Internet Technologies and Systems, March 2003.

[4] Muhammad Asif Jan, Fahd Ali Zahid, Mohammad Moazam Fraz, and Arshad Ali. “Exploiting peer group concept for adaptive and highly available services”. In Computing in High Energy and Nuclear Physics, La Jolla California, 24-28 March 2003.

[5] Sentinel Project. URL: http://sentinel.jxta.org, June 2006.

[6] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman. “An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems”. In Proc. of International Conference on Distributed Computing System, 1999.

[7] Roberto BALDONI, Fabio ZITO. “Designing a Service of Failure Detection in Asynchronous Distributed Systems”. IEEE, ISBN 0-7695-1089-2/01, 2001.

[8] Zupeng Li, Yuguo Dong, Lei Zhuang, and Jianhua Huang. “Implementation of Secure Peer Group in Peer-to-Peer Network”. In Proc. of ICCT, 2003.

[9] M. Steiner, G. Tsudik, and M. Waidner. “Key agreement in dynamic peer groups”. IEEE Transactions on Parallel and Distributed Systems, Volume:11, Number:8, pp.769-780, August 2000.

[10] Yongdae Kim, Daniele Mazzocchi, and Gene Tsudik. “Admission Control in Peer Groups”. In IEEE NCA, 2003.

[11] W. Vogels and C. Re. “WS-Membership - Failure Management in a Web-Services World”. In Intl. World Wide WebConference (WWW), 2003.

[12] R. van Renesse, Y. Minsky, and M. Hayden. “A Gossip-Style Failure Detection Service”. In Proc. Of Middleware’98, pp.55-70, IFIP, The Lake District, UK, 1998.

[13] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. “Chord: A Scalable Peertopeer Lookup Service for Internet Applications”. ACM, ISBN 1581134118/01/0008, 2001.

[14] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. “A Scalable Content-Addressable Network”. Proc. of SIGCOMM’01 ACM Conference on Applications, Technologies Architectures, and Protocols for Computer Communication, August 2001.

[15] Antony Rowstron1 and Peter Druschel. “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems”. In Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001). Heidelberg, Germany, November 2001.

[16] Ryan Huebsch, Brent Chun, Joseph M. Hellerstein, Boon Thau Loo, Petros Maniatis, Timothy Roscoe, Scott Shenker, Ion Stoica and Aydan R. Yumerefendi. “The Architecture of PIER: an Internet-Scale Query Processor”. In Proc. of the 2005 CIDR Conference, 2005.

[17] Antony Rowstron1, Anne-Marie Kermarrec1, Miguel Castro1, and Peter Druschel. “SCRIBE: The design of a large-scale event notification infrastructure”. In Proc. of 3rd International Workshop on Networked Group Communication (NGC2001), UCL, London, UK, November 2001.

[18] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, James W. O'Toole. “Overcast: Reliable Multicasting with an Overlay Network”. Proc. of 4th Symposium on Operating System Design and Implementation, San Diego, 2000.

[19] The Peer-to-Peer Trusted Library Project. URL: http://sourceforge.net/projects/ptptl/, June 2006.

[20] P. McDaniel, A. Prakash, and P. Honeyman. “Antigone: A flexible framework for secure group communication”. In Proc. of 8th USENIX Security Symposium, pages 99–114, Aug. 1999.

[21] DELIS Project. “Requirements for a P2P platform for dynamic management of large scale networks”. Deliverable D2.3.1, February 2005.

[22] The Project JXTA: P2P Framework. URL: http://www.jxta.org/, May 2006.

[23] Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, and Bill Yeager. “Project JXTA 2.0 Super-Peer Virtual Network”. Sun Microsystems Inc., URL: www.jxta.org/docs/, May 2006.

[24] The Jini Community, Sun Microsystems, URL: www.jini.org/, May 2006.

Yazılım Mimarisinde Kalite

Yazılım Mimarisinin Evrimi Üzerine Bir Durum Çalışması: Elektronik İmza Uygulamasının Sürekliliği

Alper Uğur, İbrahim Soğukpınar
Gebze Yüksek Teknoloji Enstitüsü, Bilgisayar Müh.liği Bölümü
augur@bilmuh.gyte.edu.tr, ispinar@bilmuh.gyte.edu.tr

Özet

Günümüzde, geliştirilen yazılımların önceden belirlenmiş gereksinimleri karşılayacak şekilde, olabilecek en az hatayla son kullanıcıya ulaştırılması yeterli olarak görülmemektedir. Ürün olarak sunulan yazılımın ileriye dönük gelişmelere kolayca uyum sağlayabilmesi, esnekliği ve ürün-yaşam süresinin uzun olması da gerekmektedir. Tüm bu özellikler, yazılımın yaşam çemberinde dönüm noktalarından biri olan bakım evresini etkileyen etkenlerdendir. Dikkatli tasarlanmamış bir yazılım uzun uğraşlar sonucunda uygulamaya konulsa bile kısa sürede ortadan kalkmaya, yerini yeni sürümlerine bırakmaya mahkum olacaktır. Bu ise yazılım üretiminde kullanılan zaman/insan kaynağının boşa harcanmış olmasına ve ileriye dönük birçok ticari kayba yol açacaktır.

Bu çalışmada, bir elektronik imza uygulaması ele alınarak, mimariye uygun olarak tasarım aşamasında alınacak erken kararların yazılım sürekliliğine katkısı ortaya konulmuştur. Bu katkılar araştırılırken yazılım mimarisinin evrim süreçleri göz önünde bulundurulmuş; sunulan mimari çözümde nesne tabanlı yazılımda birçok yan getirileri olan tasarım kalıplarından faydalanılmıştır.

1. Giriş

Yazılım mimarisi, karmaşık sistemlerde birbiriyle ilişkili birçok bileşeni bir araya getirerek bunların düzenli çalışmasını sağlayabilecek, yazılım üretiminin en kritik noktalarından biridir. İyi bir mimari, yazılımın ölçeklenebilirlik, performans, dayanıklılık, uyumluluk gibi temel gereksinimlerini yerine getirmesini sağlarken; kötü tasarlanmış mimari bir felakete yol açabilir [1].

Yazılım mimarisi bir yazılımın uzun dönemdeki başarısının temelini oluşturmaktadır. Yazılımın başarısı; onun yaşam süresi, sahadaki kararlılığı, değişime yatkınlığı ve maliyet/ürün oranı gibi yetkinlikleri ile orantılıdır [2].

Teknolojideki hızlı gelişmeler, ancak üretilen yazılım bu teknolojilerle birlikte evrimleşebildiği sürece yazılıma bir artı sağlar. Bu da genellikle müşteri tarafına, ürüne uyarlanmış faydalı uygulamalar olarak dönecek ve yazılımın ürün ömrünü arttıracaktır.

Bu faydalarıyla birlikte teknolojideki hızlı gelişmeler, kimi zaman da mimaride öngörülmemiş eksikliklerden dolayı yazılımın kararlılığını azaltabilir veya ürün aşamasına gelmiş bir yazılımın planlanan ömrünü kısaltabilir.

Genel yapılan hatalardan biri acil müdahalelere önem verip uzun-dönem çalışmasının tamamen ihmal edilmesidir. Buna sebep olarak; nelerin tam olarak değişebileceğinin bilinememesi, yeni gereksinim analizi sürdüğü sürece üretimde analiz aşamasında kalınacağı, bunu gerçekleştirmek için yeterli bütçenin, yeterli zamanın olmaması öne sürülmektedir [3].

Bir mimari ürün yetkinliklerine göre yeniden gözden geçirilip yukarıda sıralanan kalite hedefleri doğrultusunda evrimleşebilir. Mimari evrim, amacı yazılım mimarisinin geçerliliğini sistematik metotlar ve yöntemler kullanılarak sağlamak olan bir aktivitedir [4]. Bu aktivitede amaçlanan sorgulanan mimari için evrim gerçekleştiğinde, yazılım için hedeflenen kalite seviye ya da seviyelerine ulaşılmış olmasıdır. Makalenin takip eden bölümlerinde, bu aktiviteye örnek olarak bir elektronik imza uygulamasının süreklilik kalite hedefine göre evrimine değinilmekte ve bunun sonuçları ortaya konmaktadır.

2. Elektronik imza uygulamasının evrimi

Bu bölümde bir elektronik imza uygulamasına ait mimarinin olası evrim süreci ele alınmıştır. Bu süreç esnasında ortaya çıkan tasarım zayıflıkları ve problemlere kısaca değinilmiş ve alınabilecek tedbirler sunulmuştur.

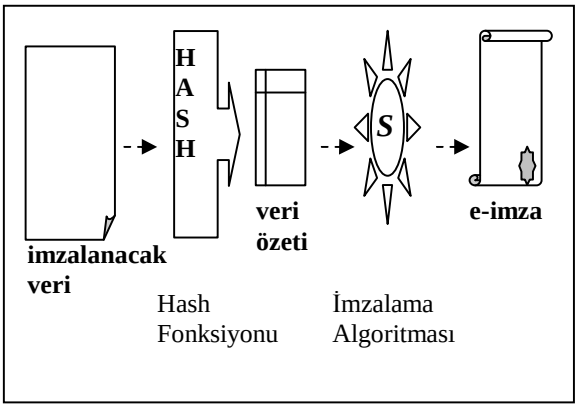
Bölüm sonunda ise temelleri oturmuş olan bir elektronik imza uygulamasına ait mimari üzerinde çalışılmaya başlanacak ve mimari evrimin

aşamalarından olan problem tanımlama ve kalite hedefinin belirlenmesine yer verilecektir.

2.1. Sorgulanan uygulama: elektronik imza

Elektronik imza, geleneksel olarak el ile atılan imzanın sayısal ortamda karşılığına ikame edebilecek uygulamadır [5].

Pratikte sayısal imzalama işlemi imzalanacak dokümanın tek yönlü bir fonksiyondan geçirilerek üzerinde işlem yapılabilecek özgün bir özetinin çıkarılması ve bu özetin uygun bir imzalama algoritmasından geçirilmesi ile gerçekleştirilir. Sayısal imzalama şeması Şekil 1'de gösterilmektedir. Bu işlemler sonucu ortaya çıkan imzaya o dokümanın elektronik imzası denilebilir.



Şekil 1. Sayısal imzalama şeması

Elektronik imza uygulamalarında Hash fonksiyonu olarak MD5, SHA–1 ve yaygın olmasa da SHA–256 kullanılırken, imzalama algoritması olarak RSA, Rabin, Schnorr, Sayısal İmza Standardı olarak da kullanılan ElGamal algoritmaları sayılabilir [6]. Çalışmanın kapsamının dağılmaması amacıyla adı geçen algoritma ve hash fonksiyonları burada daha fazla detaylandırılmamıştır.

2.2. İlkelden gelişmişe elektronik imza tasarımı

İlkel elektronik imza uygulaması var olan problemin çözümüne yönelik tasarlanmıştır. Bu aşamada problem, istemci uygulamanın işlevine göre belirlenmiş ve herhangi bir metnin ya da veri üzerinde sayısal imza şemasının gerçeklenerek elektronik imzanın ortaya çıkarılması olarak ele alınmıştır.

Oldukça ilkel bir tasarımı algoritmaların birer metot olarak uygulamaya gömüldüğü yapıyı ele alarak inceleyebiliriz. Bu tasarım türü kapalı yazılımı temsil etmektedir. Yazılımda gerçeklenmesi muhtemel herhangi bir gözden geçirme ya da değişiklik tüm uygulamayı etkileyecek ve uygulamanın bu işlemler süresince devre dışı kalmasına sebep olacaktır.

Aynı zamanda bu tasarım, olası yapısal güvenlik açıklıklarına karşı dayanıklı olarak geliştirilen kriptografik algoritmaları kodlama zayıflıklarından doğacak saldırıların karşısında korunmasız bırakacaktır. Bu durumda elektronik imza uygulaması çalışır göründüğü halde temel güvenlik işlevini aksatacak demektir.

Bunu engellemek yolunda alınacak ilk önlem yapıdan tamamen vazgeçip modüler sisteme uyum olacaktır. Böylelikle küçük değişiklikler tüm uygulamayı etkilemeden yapılabilecektir. Modüler sisteme geçiş sürecine mimaride arayüz ve işlevin birbirinden ayrılması ile başlanıp bu süreç bir sonraki adımda ise işlevselliği oluşturan temel metotların tanımlanarak uygulamadan bağımsızlaştırılmasıyla sürdürülecektir. Verilen bu kararın alınmasındaki en büyük etken yukarıda da bahsedilen, yazılımın üretimine etkiyen karmaşıklık, uyumluluk, değişkenlik gibi temel etkenlerdir[7].

Daha önceki aşamaların her birinde tamamen değişen yazılım mimarisinin, bu aşama ve sonrasında, artık sadece temel etkenler doğrultusunda geliştirilecek eklentiler ve yapılan iyileştirmeler ile yoluna devam edeceği gözlenecektir. Bu süreçte yazılım mimarisi belirlenen ya da öngörülen açıklık ya da zayıflıklara odaklanılarak, bunlar sonucunda planlanacak kalite hedeflerini yakalayacak şekilde evrimleşecektir.

2.3. Kalite hedefi: süreklilik

Yazılım mimarisinin evrimleşmesini tetikleyen etkenlerden en önemlisi gereksinimlerin iyi analizidir. Bu şekilde, yazılım gereksinimleri tespit edilerek mimaride bir kalite hedefi ya da hedefleri belirlenecek ve bu hedeflere ulaşmayı sağlayacak yapılar mimariye dahil edilerek evrimleşme aktivitesi sürdürülecektir.

Bu çalışmada kalite hedefinin belirlenmesine yazılım mimarisinde hangi yapının önem arz ettiği göz önünde bulundurularak başlanmıştır. Elektronik imza uygulamasında uygulamanın kimlik denetimi, veri bütünlüğü, yetkilendirme gibi temel işlevlerinin yerine getirilmesini uygulamada kullanılan kriptografik algoritmaların sağladığı bilinmektedir.

Önceki bölümde teknolojideki hızlı gelişmelerin kimi zaman yazılıma artı etkileri olduğundan kimi zamansa mimarideki eksikliklerden kaynaklanabilecek olumsuz yan etkileri olabileceğine değinilmişti.

Elektronik imza uygulamaları içerdiği algoritmik yöntemlerin sağladıkları güvenlik seviyelerinin

zamanla değişkenliğinden dolayı gelişen teknolojiyle paralel olarak gelişmek zorundadırlar.

Yapılan çalışmalar göstermiştir ki kullanılan kriptografik algoritmaların anahtar uzunluklarının değiştirilerek problem uzayını büyütmesi ve böylelikle hızla gelişen teknolojiye bir süre daha dayanacağı öne sürülse de [8] hash fonksiyonları için aynı şey söz konusu değildir.

Birçok sertifika uygulamasında kullanılan MD5 sıkıştırma fonksiyonunun çarpışma saldırısına maruz kaldığında yetersiz olduğunu gösteren ve hem MD5 hem de SHA–1 hash fonksiyonlarının iyi bir hash fonksiyonunun gereksinimlerinden olan farklı verilerin aynı çıktıyı oluşturmaması özelliğini sarsan çalışmalar yapılmıştır [9][10].

Bu sonuçlar kullanılan algoritmaların uygulama sürecinde dinamik yapıda olmaları gerektiğini ortaya koymaktadır. Orta ve yüksek güvenlik seviyeli elektronik imza uygulamalarında kullanılan Hash fonksiyonlarının kısa sürede ve kriptografik algoritmaların uzun dönemde yenileriyle değiştirilmesi gerektiği ve değişimin teknolojinin sürekli gelişmesiyle birlikte devam edeceği de açıktır [11][12].

Bu bilgilere dayanarak kullandığı metotlar, kriptografik algoritmalar ve fonksiyonlar olan elektronik imza uygulamalarında sürekliliği sağlayacak ve mimarinin yazılımın ürün-yaşam ömrünü arttıracak şekilde evrimini sağlamanın gerekliliği öne sürülebilir. Bu durumda alınacak ilk tedbir geleneksel yazılım yöntemleri bırakılarak yazılımın adı geçen algoritma ve fonksiyonlarla ilişkisinin tamamen modüler olarak tasarlanması ve bu şekilde uygulama ortamına sürülmesidir. Böylelikle elektronik imza uygulaması gelişen teknolojiye ayak uydurabilecek, sürekli yeniden inşa gerektirmeyerek çok küçük değişikliklerle hizmet vermeye kesintiye uğramaksızın devam edecektir.

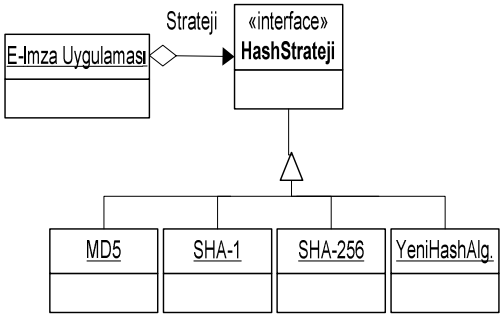
3. Tasarımın evrimleşmesinde kullanılan metot: strateji tasarım kalıbı

Önceki bölümde, mimari evrim, tanımında sistematik metotlar kullanılarak gerçekleştirilen bir aktivite olduğu belirtilmişti. Bu aktivitede ilk adım olarak yazılım mimarisinde ortaya çıkabilecek yazılımın yaşam ömrü sorununa karşı gözden geçirilecek kalite hedefi olarak süreklilik seçildi.

Kalite hedefine ulaşılırken izlenecek yol hangi unsurlar yeniden tasarım yapmayı gerektirir anlayışından çok, tasarımı tümüyle yenilemeden nelerin değiştirilebilineceğine yönelik olmuştur [13].

Uygulanabilecek birçok tasarım yönteminin yanı sıra bu hedef, temelde elektronik imza uygulamalarında, algoritma bağımsızlığı sağlayabilecek bir yapı gerektirmektedir. Bu aşamada GOF[1]'un tasarım kalıpları katalogunda [13] *davranışsal kalıp* sınıflandırmasının altında yer alan *strateji tasarım kalıbı*nın kullanılmasını işlevsel olarak uygun bulundu.

Strateji kalıbı, aynı aile üyesi ve birbiriyle ilişkili her bir algoritmanın ortak bir süper-sınıfa ait farklı alt-sınıfların strateji nesneleri olarak tanımlanabilmesini ve kapsüllenmesini sağlar. Uyarlanacak tasarımda, algoritmayı kullanacak nesnenin ilgili strateji nesnesine bağlanması yeterli olacaktır. Bununla beraber kurulacak ilişkide, algoritmadaki herhangi bir değişiklik uygulamayı ya da istemci programı etkilemeyecektir [14]. Bu kalıbın yapısı Şekil 2'de sunulmuştur.



Şekil 2. Strateji kalıbının yapısı

Şekil 3'te ise uygulamadaki hash işlemlerinden kod parçaları verilmektedir. İlk parça yazılımın hedef belirlendiğindeki halini sonraki kod parçası ise yazılımın strateji kalıbı uyarlanmış halini göstermektedir.

Burada yer verilen kod parçalarından önceki durumda yeni algoritma sisteme eklendiğinde istemci uygulamanın uyum sürecinin zahmetli olacağı gözlemlenebilir. Tüm kod baştan sona gözden geçirilerek gereken yerlerin değiştirilmesi ve kod satırlarının eklenmesi gerekecektir.

Oysa tasarım evrimleştikten sonraki kodlar incelendiğinde mimari gereği yeni algoritmanın tak çalıştır kolaylığıyla yazılıma dahil edilmesinin mümkün olduğu görülmektedir.

Bu yapıda;

- Strateji, mimaride desteklenen tüm algoritmalar için bir arabirim (interface) tanımlar. Uygulama bu arabirimi kullanarak somut-stratejilere erişir. Somut-stratejiler bu mimaride şu an kullanılan belli başlı hash fonksiyonları ve olası

[1] Gang of Four takma adıyla bilinen yazarlar: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

kullanılabilecek Yeni Hash Algoritması fonksiyonu ile sınırlandırılmıştır.

- Somut-stratejiler, arabirimi kullanan algoritmaları gerçekleştirmektedirler.
- İstemci uygulama ise somut-strateji ile yapılandırılmış, strateji nesnelerine referans yoluyla bağlantı temin edilmiş ve strateji nesnesinin kullanılacak verilere erişimi için arabirim tanımlanmıştır.

```
class MD5 {
      byte [] hashing (byte [] msj) {
        ..//MD5'e özgün işlemler
      }
}
class SHA1 {
      byte [] hashing (byte [] msj) {
        .. //SHA1'e özgün işlemler
      }
}
static byte [] hashingOnceki (byte [] msj)
{  ...   //case sha:
   SHA1 S = new SHA1 ();
   SHAHash=S.hashing (msj);
   ...   //case md5:
   MD5 M = new MD5 ();
   MD5Hash=M.hashing (msj);
   ...
}

class MD5 implements HashStrateji {
      byte [] hashing (byte [] msj) {
        ..//MD5'e özgün işlemler
      }
}
class SHA1 implements HashStrateji {
      byte [] hashing (byte [] msj) {
        .. //SHA1'e özgün işlemler
      }
}
interface HashStrateji {
      byte [] hashing (byte [] msj);
//algoritma bağımsızlığı sağlandı
}
static byte [] hashingSonraki (byte []msj)
{ ...
  secilenHashStrateji.hashing (msj);
}
```

Şekil 3. Örnek yapı

Bu algoritmaların çıktıları değişken olsa bile uygulamanın arabirimle ilişkisinde kullanılan algoritma girdilerinin benzer olmaları, farklı algoritmaların strateji kalıbı ile soyutlanmasını kolaylaştırmıştır.

İstemci uygulamadan algoritmanın soyutlanması elektronik imza uygulamaları için tercih edilecek bir başka güvenlik özelliğidir. Böylece tasarım esnasında uygulama karmaşıklığının azaltılmasının yanı sıra kriptografik algoritmaların inşasında gerçekleşebilecek olası kodlama zayıflıkları da uygulamadan soyutlanmaktadır.

Bununla birlikte satın alınmış, güvenlik testlerinden geçmiş algoritma modüllerinin uygulamaya uyarlanması, mimariye bir somut-strateji nesnesi tanımı eklenerek tamamlanacak kadar kolaylaşacaktır. Böylece bağımsız analiz gerektiren kendine özgü güvenlik testleri olan kriptografik algoritmaların kodlama karmaşıklığı da asgari seviyeye çekilebilecektir.

Somut-stratejilere arabirim ile ulaşılması, uygulamanın, farklı ihtiyaçlara göre değişken güvenlik seviyelerine sahip kriptografik algoritmalar arasında seçim yapabilecek şekilde yapılandırılmasını kolaylaştıracaktır.

Strateji kalıbının uygulanmasından dolayı ortaya çıkabilecek nesneler arası haberleşme artışı bu evrimin kusuru olarak belirtilebilir.

4. Sonuç ve öneriler

Bu çalışmada bir yazılım mimarisinin evrimi, elektronik imza uygulamasındaki süreklilik probleminin çözümü çerçevesinde incelenmiştir.

Mimari evriminde kalite hedefi seçilirken uygulamaya özgün sorunlar ortaya konmuş ve bu sorunlara yönelik esnek bir düzenlemeye gidilmiştir. Elektronik imza uygulamasında sürekliliği etkileyen ve uygulamanın temel taşlarından olan algoritmaların esnekliğine yönelik çalışmalar yapılması gerekliliği ortaya konmuştur.

Bu ve devam eden aşamalarda ise tasarımın tamamen yenilenmesi yerine evrimin de özü olan kısmi olumlu değişikliği sağlamak üzere strateji tasarım kalıbından yararlanılmıştır. Bu kalıbın mimariye dahil edilmesi ile algoritmalar uygulamadan soyutlanmış ve gereken esneklik sağlanmıştır.

Yazılım mimarisinin gözden geçirilmesine ve olası yeni gereksinimler karşısında tekrar tasarlanması için yeterli zaman ve belirsizliklerin çokluğu görüşlerinin aksine, var olan mimarinin ileriye dönük gereksinimler için evrimleşmesi ile olası sorunların da çözümlendiği gözlemlenebilir. Bu çalışmada da süreklilik hedefi ile evrimleşen mimarinin aynı zamanda ürün güvenliğini de arttırdığı görülmüştür.

Ortaya konan yeni evrimleşmiş mimari yapı daha önce faydalanılamayan üretim/satın alma dengesini de değiştirecek türden olmuştur. Geleneksel yöntemdeki bütün algoritmaların gömülü olarak uygulamaya eklenmesi zorluğuna karşı, bu mimari, satın alınılacak, arabirim referansları hazır algoritmaların, uygulamada kullanılabilmesinin de önünü açmıştır. Bu durumun analizi çalışmanın kapsamı dışında bırakılmış olsa da böylelikle ileriye dönük yapılan gereksinim analizi ile kaybedildiği düşünülen zaman, hazır modüllerin

uygulamaya dahil edilmesiyle telafi edilebilineceği açıktır.

Evrimin nesneler arası haberleşmeyi artıracak şekilde gelişmesi gözlenen kusurlardan biridir. Ama bu kusurun etkileri, uygulamaya eklenecek değişken güvenlik seviyelerine sahip algoritma seçimini sağlayan bir karar mekanizması ile asgari seviyeye indirilebilinir.

Strateji kalıbının uygulamanın farklı ihtiyaçlara göre değişken güvenlik seviyelerine sahip algoritma seçimini yapabilecek şekilde evrimleşmesi mimarinin başka bir faydası olarak belirtilebilir. Böylece düşük güvenlik seviyesi gerektiren işlemlerde diğerlerine nazaran az işlem ya da hafıza gerektiren algoritmanın seçimi yapılabilecek ve yazılımın kararlılığı arttırılmış olacaktır.

Bu mimariyle birlikte ortaya çıkan yazılımın birim testlerinin de oluşturulan modüler yapıdan dolayı daha kolay yapılacağı göz önünde bulundurulmalıdır. Bu bağlamda yazılım mimarisinde yapılacak olumlu bir değişikliğin yaşam çemberinde tasarımdan teste ve ürün-yaşam sürecine tüm evreleri olumlu biçimde tetikleyeceği söylenebilir.

Kaynakça

[1] Garlan D., "Software Architecture: a Roadmap" *The Future of Software Engineering*, Anthony Finkelstein (Ed.), ACM Press, 2000

[2] Hohmann L., *Beyond Software Architecture Creating and Sustaining Winning Solutions*, Addison Wesley, 2002

[3] Shalloway A., Trott J.R., *Design Patterns Explained A New Perspective on Object-Oriented Design*, Second Edition, Addison Wesley,2004

[4] Bahsoon R., Emmerich W., "Evaluating Software Architectures: Development, Stability, and Evolution", *Proceedings of ACS/IEEE Int. Conf. on Computer Systems and Applications*, Tunus, 2003

[5] Mason S., "Electronic Signatures: The Technical and Legal Ramifications", *Computers and Law*, Dec 1999/ Jan 2000, Volume 10, Issue 5,37- 44,39, 2000

[6] Mao W., *Modern Cryptography Theory and Practice*, *Prentice Hall*, New Jersey, 2004

[7] Budgen D., *Software Design*, Addison Wesley, 2003

[8] Kaliski B., "TWIRL and RSA Key Size", *RSA Laboratories Technical Notes and Reports*, Mayıs 2003

[9] Cid C., "Recent Developments in Cryptographic Hash Functions: Security Implications and Future Directions", *Information Security Technical Report*, Volume 11, Issue 2, 100-107, 2006

[10] Zaba S., "Digital signature legislation: The first 10 years", *Information Security Technical Report*, Volume 11, Issue 1, 18-25, 2006

[11] Landau S., "Find me a hash", *Notices of the AMS*, 53(3):330–2, Mart 2006

[12] ECRYPT, "Recent collision attacks on hash functions: ECRYPT position paper", *ECRYPT Network of Excellence Report 2004*, <http://www.ecrypt.eu.org/documents/STVL-ERICS-2-HASH_STMT-1.1.pdf> , Ekim 2006.

[13] Gamma E., Helm R., Johnson R., Vlissides J, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1995.

[14] Kuchana P., *Software Architecture Design Patterns in Java*, CRC Press, USA, 2004

Evaluating Software Architectures for Successful Software Projects

Burak Turhan
Boğaziçi University
turhanb@boun.edu.tr

Ataç Deniz Oral
Boğaziçi University
atacdeniz.oral@siemens.com

Ayşe Bener
Boğaziçi University
bener@boun.edu.tr

Abstract

A successful software project needs to satisfy the customer's needs and should be completed within schedule and budget. Ensuring quality standards in different phases of software development lifecycle brings along the satisfaction of key areas for success.

To measure and evaluate software quality, quantitative metrics should be collected and processed at each step. Successive steps should be taken only if the quality expectations of the current step are satisfied in pre-defined confidence levels.

This work focuses on the methods for detecting faults in the architectural level of software development. For this purpose, a brief survey on proven software architecture evaluation methods is given. Furthermore, we discussed that the metrics which are adapted from object-oriented design can be collected in the architectural level of software development. We conclude that a comparative analysis can be carried out with these metrics for fault prediction using different statistical methods.

1. Introduction

A software project needs to have two important properties in order to be classified as successful. The first and the obvious one is that the final product should satisfy the customer's expectations. Clearly, this is not enough for a software company to permanently hold a share in a competitive market place. Additional functionality and originality should also be involved. But, software companies and project managers have realized a long time ago that producing software with better and more complex functionality does not always yield successful projects in terms of customer satisfaction; due to quality problems.

The second important property of a successful software project is that it should be completed within schedule and budget. Most of the software projects are classified as unsuccessful by just using this second property [1]. To satisfy the schedule constraints, every step of the software lifecycle should be carefully planned and employed. Moving on to the next step requires the most possible confidence on the quality of the preceding level. The budget constraint depends on both satisfying the schedule constraint and the quality of the final product. Previous research has shown that fixing faults in a software system after it has been delivered to the customer is a lot more expensive than finding and fixing it during the requirements and design phases, which means just after they are introduced [2], [3]. Unfortunately, most software companies do not take maintenance costs into account and try to minimize the expenses considering only the period up to the release of the software product.

As most researches point out, the key element that affects the success of a software project is the concept of quality [4], [5]. Ensuring to have quality standards at each step of software development lifecycle automatically introduces the satisfaction of customer's expectations, schedule constraints and the budget constraints, brining success along.

Software quality is a hot topic that has attracted the attention of many researchers [4], [5]. The number of faults in a software system is considered as an indicator of software quality. For the sake of empirical validation, software quality researches need to extract information from software in quantitative forms. Thus, several metrics that are assumed to represent the characteristics of software are proposed [6], [7], [8], [9], [10]. Metrics enable us to measure and quantify the properties of software systems.

One of the greatest problems that developers are facing nowadays is the defects introduced to the software products at every stage of the software development lifecycle. It is generally thought to be best to collect these metrics at later stages of software projects (i.e. coding) since in that case they would be better indicators of the properties being investigated.

However, as we pointed out earlier, detecting and correcting defects at earlier stages of software development is more cost effective [2]. This fact shifted the trend for collecting the defects only after the software system is implemented to evaluate the system, and also to allocate the testing resources; to collecting metrics at earlier stages of the lifecycle such as requirements and architectural design.

Architectural design level is the core abstraction of a software project that the successive levels are built on. Ensuring the correctness of software architecture helps developers to save a considerable amount of effort compared to detecting a fault in the architectural level in later phases and returning to that point. Thus, we emphasize the importance of high quality architecture.

In this article, we focus on the metrics defined to be collected at the architectural level. We survey the studies on methods for software architecture evaluation and adapting object-oriented metrics for software architecture quality. We believe the serious consideration of these methodologies will minimize the problems originating from architectural decisions. Section 2 briefly explains the methods of architecture evaluation. In Section 3 we give background information about object-oriented metrics that can be adapted for evaluation of software architecture. Finally, the conclusion is given in Section 4.

2. Methods for Software Architecture Evaluation

In this section we give brief descriptions of several software architecture evaluation methods. These are: Software Architecture Analysis Method (SAAM) [11], Architecture Trade-off Analysis Method (ATAM) [12] and Cost Benefit Analysis Method (CBAM) [13].

2.1 Software Architecture Analysis Method

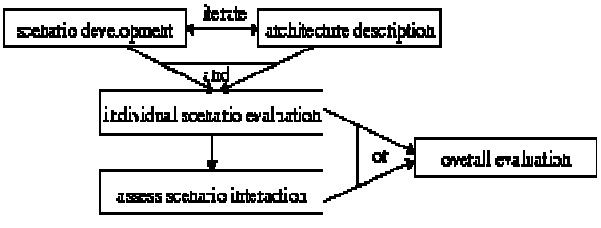


Figure 1: Activities and dependencies in scenario-based analysis (adapted from [14])

Software Architecture Analysis Method (SAAM) is a scenario based method to evaluate certain quality attributes of software architecture. These attributes include easy modification and smooth integration of the programs as well as creating software programs that are portable, extensible and accurate in their functionality [11]. The method is summarized in Figure 1. At the first step, scenarios that the system should support are developed. In parallel, possible architectures are described in a natural or formal language. The third step is to evaluate each scenario by generating the responses that the system should give. If conflicting scenarios exist, the architecture is modified in the next step. In the final evaluation step, each scenario is given a weight proportional to their importance and the overall performance of the architecture is evaluated. Different architectures may be evaluated with these scenarios and the most appropriate one is chosen.

The definition of scenarios to be evaluated needs strong communication among stakeholders. This helps the stakeholders to understand the architecture in depth [15]. But this also causes the scenarios to be limited with the stakeholders' capabilities and experiences. Furthermore, the weight assignment to scenarios should be handled carefully in an objective manner.

Two extensions to this method are SAAMCS (SAAM Founded on Complex Scenarios) and SAAMER (SAAM for Evolution and Reusability) [16]. SAAMCS assumes that the complexity of scenarios is the most important factor to assess risks whereas SAAMER approach, as the name suggests, is to focus on the evolution and reusability attributes [16].

2.2 Architecture Trade-off Analysis Method

Architecture Trade-off Analysis Method (ATAM) is also a scenario based method to evaluate certain quality attributes of software architecture. Similar to SAAM, quality attributes of ATAM focus on easy modification, integration, portability and extensibility [12]. One advantage over SAAM is that ATAM can reveal the dependencies and trade-offs of attributes, as the name suggests.

ATAM has four main steps, namely: presentation, investigation and analysis, testing, and reporting [12]. In the presentation step different stakeholders explain their view on the architecture. In the investigation and analysis step the trade-off about the quality attributes and the architecture is discussed. The testing step ensures the previous iterative step satisfies all stakeholders' needs up to an acceptable level. And in the last step, the reporting step, the architecture and the obtained results are summarized.

It is reported that ATAM enables stakeholders to understand the architecture more clearly and yields a

better communication, and the architecture document is improved [15].

2.3 Cost Benefit Analysis Method

Cost Benefit Analysis Method (CBAM) takes cost and schedule constraints into consideration while evaluating the architecture. Different than SAAM and ATAM, CBAM adds cost, benefits and risks as additional quality constraints [13]. In CBAM, different appropriate architectures are proposed for each scenario and considering the needs of the stakeholders and the costs, a decision on the final architecture is given [5].

2.4 Other Methods

Other known methods about software architecture evaluation include Architecture-Level Modifiability Analysis (ALMA) [17], Family-Architecture Analysis Method (FAAM) and Active Reviews for Intermediate Designs (ARID) [15]. Interested readers are to consult [15], for a detailed explanation and comparison of these methods.

3. Evaluating Object-Oriented Design

3.1 Metrics for Object-Oriented Design

After the object-oriented software development paradigm became popular, a lot of research has been dedicated to finding appropriate metrics especially for object-oriented design [6], [7], [18], [19].

Given the demand for higher quality software a number of metrics have been proposed for object-oriented approach [6], [7], [18], [19].

Studies about adapting object-oriented metrics for the quality evaluation of software architecture [18], [19] encouraged the authors to give a summary of the most popular object-oriented metrics. Some of these metrics are listed below:

- WMC (Weighted Methods per Class): In a simple framework, where all methods in a class are considered to be equally complex, this metric is equal to the number of methods in a class.
- DIT (Depth of Inheritance Tree): This metric is defined as the maximum depth of inheritance graph of each class.
- NOC (Number of Children of a Class): This metric is equal to the number of direct descendants of a class for each class.
- CBO (Coupling between Object Classes): A class is considered to be coupled to another class if it uses methods and/or instance variables of that other class.
- RFC (Response for a Class): This metric quantifies the number of methods that can be executed for a message received by an object of a class.
- LCOM (Lack of Cohesion in Methods): This metric is equal to the number of pairs of methods of a class that share instance variables, subtracted from pairs of methods of the same class that do not share instance variables. This value is taken to be zero whenever the above operation results in nonnegative values.

Some hypotheses were given for these metrics [20]. For the WMC metric the hypotheses is that, a class with higher values for WMC will be more complex than classes with lower WMC values, and thus will be more fault-prone. For the DIT metric, again higher values are indicative of being more fault-prone. This is because of the greater number of definitions inherited from the ancestors. The NOC metric does not have any implications on complexity. However, having more children means that modifications to classes of this kind will be harder, because of the possible impacts on its children. This means that if NOC metric for a class is high it should be carefully tested. The CBO metric is what a designer would want to minimize as far as possible during the design process. This is because classes with high CBO values will depend on other classes more than classes with low CBO values. The RFC metric is an indicator of the complexity of the functionality implemented in the class. This means that classes with higher RFC metrics will be more fault-prone. LCOM is like CBO, it is a value a designer would want to minimize during the design. This is because the preference in object-orientation the classes should be as highly cohesive as possible. Higher LCOM values is an indication of weak design since, high values mean that functions that are not related (to the same set of data), is encapsulated together.

3.2 Validation of the Metrics for Object-Oriented Design

The hypotheses above (except for NOC) were tested by various researchers [20], [21]. Basili et al. used some metrics gathered from software products developed by students, and Gyimothy et al. tested the hypotheses using metrics of open source software. The results obtained were very successful, meaning the above metrics were very successful at capturing the

complexity of the classes and therefore predicting the fault-proneness of these building blocks.

The metrics used by Basili et al., and Gyimothy et al. were collected from projects that were already implemented [20], [21]. Therefore in both researches authors use code analyzers to extract the aforementioned metrics. Once the metrics were extracted authors used logistic regression, linear regression, and machine learning techniques to empirically validate the metrics. All of the metrics except for NOC were found to have a very significant relationship with fault-proneness. In the case of NOC (Number Of Children), not surprisingly this metric turned out to be large for classes that are heavily reused, and since reuse was observed as a significant factor that decreases fault density, having large number of children did not indicate a higher fault-proneness [20].

3.3 Implications of the Metrics on the Future of Software Design

Despite the small number of metrics proposed so far, contemporary research on object-oriented design metrics focuses heavily on quantifying the level of coupling and cohesion of object-oriented classes [22], [23]. In [22] authors focus on investigating the variety of cohesion metrics proposed so far, by considering dependent instance variables (instance variables whose values are computed using other instance variables). In [23] authors try to quantify the level of coupling and cohesion in the design using Wood's Task Complexity Model. By this way the research aims identifying the effect of the complexity of software on maintainability and hence the cost of the project. At this point it would not be too bold to say that, the future of software design will also be heavily directed by efforts for reducing its complexity, and therefore reducing the costs and increasing quality.

As pointed out by [20] and as seen to be left out in [21], [22], [23], one point that is yet to be researched would be metrics that are language specific. This might enable the designers to identify a more comprehensive set of attributes of their design, and might result in a better evaluation of the design.

5. Conclusion

Success in software projects are defined as satisfying customer's needs within schedule and budget. With this in mind, it is discussed that the key element to achieve successful software projects is to obtain a quality level at each step of the software development lifecycle. The earlier defects are detected and a quality level is reached, the more cost effective the software projects become. The focus of the study is on the architectural level evaluation and this article presented proven methods and applicable metrics to evaluate the quality of software architecture.

We believe that these methods and metrics are invaluable sources of supplementary information for designers when evaluating their design, and at the same time these can guide managers in allocating project resources more appropriately since their ultimate goal is to create high quality, low cost software.

Going forward research in this area should employ analysis of metrics extracted at design time. One possibility would be to use a wide range of machine learning techniques for fault prediction and to compare results with the existing research. It would also be possible to compare the accuracies of the design metrics proposed until now, and at the same time figure out the best means of using them by making a comparative analysis of the outcomes of various machine learning techniques.

Acknowledgements

This research is supported in part by Boğaziçi University research fund under grant number BAP-06HA104.

References

[1] "CHAOS Chronicles, 2004 Third Quarter Research Report", 2004.

[2] B. Boehm and V.R. Basili, "Software Defect Reduction Top 10 List", IEEE Computer, Vol. 34, No. 1, January 2001, pp. 135-137.

[3] Abowd, G., Bass, L., Clements, P., Kazman, R., Northrop, L., Zaremski, A., "Recommended best industrial practice for software architecture evaluation," Software Engineering Institute, CMU/SEI-96-TR-025, 1996.

[4] Barbacci, M.R., et al. "Quality Attributes", Software Engineering Institute, CMU/SEI-95-TR-021, 1995.

[5] Boehm, B. et al. "Characteristics of Software Quality", New York: Elsevier North-Holland Publishing Company, Inc., 1978.

[6] F.B. Abreu and R. Carapuça, “Candidate Metrics for Object-Oriented Software within a Taxonomy Framework”, J. System and Software, vol 26, no. 1, Jan. 1994, pp. 87-96.

[7] S.R. Chidamber and C.K. Kemerer, “A Metrics Suite for Object-Oriented Design”, IEEE Tans. Software Engineering, vol. 20, no. 6, June 1994, pp. 476-493.

[8] Boehm, B. “Software Engineering Economics” Prentice Hall, 1981

[9] B.Barry, C.Bradford, H.Ellis, W.Chris, M.Ray, S.Richard, “The Cocomo II Software Cost Estimation model”, International Society of Parametric Analysts, 1995

[10] Aoyama M., “Metrics and Analysis of Software Architecture Evolution with Discontinuity”. In Aoyama M et al (eds.), Proc. Intl. Workshop on Principles of Software Evolution, IWPSE 2002, May 19 -- 20, Orlando, FL., pp. 103 - 107

[11] Rick Kazman, Len Bass, Gregory Abowd, and Mike Webb, "SAAM: A Method for Analyzing the Properties Software Architectures," Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May 1994, pp. 81-90.

[12] “ATAM: Method for architecture evaluation”: ATAM - Architecture Trade-off Analysis Method report: http://www.sei.cmu.edu/ata/ata_method.html

[13] “CBAM: Cost Benefit Analysis Method”, http://www.sei.cmu.edu/ata/products_services/cbam.html

[14] Rick Kazman, Gregory Abowd, Len Bass, Paul Clements, "Scenario-Based Analysis of Software Architecture",http://www.sei.cmu.edu/architecture/scenario_paper/ieee-sw3.htm

[15] Mugurel T. Ionita, Dieter K. Hammer and Henk Obbink, “Scenario-Based Software Architecture Evaluation Methods: An Overview”, Workshop on Methods and Techniques for Software Architecture Review and Assessment at the International Conference on Software Engineering, Orlando, Florida, USA, May 2002.

[16] L. Dobrica and E. Niemela, “A Survey on Software Architechture Analysis Methods”, IEEE Trans. Software Engineering, vol. 28, no. 7, July 2002, pp. 638-653.

[17] Bengtsson, PerOlof, Ph.D. Thesis, “Architecture - Level Modifiability Analysis”, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Sweden 2002.

[18] Dumke, R.; Zuse, H., “Software Metrics in Object-Oriented Software Development” in: Lehner, F.: Die Wartung von wissensbasierten Systemen. Haensel-Hohenhausen Publ., Frankfurt, Washington, 1994, pp. 58-96

[19] Gillibrand, D.; Liu, K., “Quality Metrics for Object-Oriented Design”, JOOP, January 1998, pp. 56-59

[20] V.R. Basili, L.C. Briand, and W.L. Melo, “A Validation of Object-Oriented Design Metrics as Quality Indicators”, IEEE Trans. Software Engineering, vol. 22, no. 10, Oct 1996, pp. 751-761.

[21] T. Gyimothy, R. Ferenc, I. Siket, “Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction”, IEEE Trans. Software Engineering, Vol. 31, No. 10, Oct. 2005, pp. 897-910.

[22] H.S. Chae, Y.R. Kwon, D.H. Bae, “Improving Cohesion Metrics for Classes by Considering Dependent Instance Variables”, IEEE Trans. Software Engineering, Vol. 30, No. 11, Nov. 2004, pp. 826-832.

[23] D.P. Darcy, C.F. Kemerer, S.A. Slaughter, J.E. Tomayko, “The Structural Complexity of Software”, IEEE Trans. Software Engineering, Vol. 31, No. 11, Nov. 2005, pp. 982-995.

Meeting Nonfunctional Requirements through Software Architecture: A Weapon System Example

Kadir Alpaslan Demir
Department of Computer Science
Naval Postgraduate School
833 Dyer Road, Monterey, CA 93943, USA
kdemir@nps.edu

Abstract

Meeting nonfunctional requirements is as important as meeting functional requirements. A well-designed software system architecture helps to ensure that the necessary quality attributes of the system are satisfied. The goal of this paper is to show how a system's software architecture can be designed to achieve its nonfunctional requirements. The development process is explained using a weapon system example named Mine Neutralization System for navy mine hunting ships. Also, a novel aspect of this paper is the introduction of a new architectural style. The style is described via an example.

1. Introduction

All software systems have a software architecture whether or not it is explicitly spelled out in the design documentation [1]. Architecture is the backbone of the software system. Therefore, it encompasses all the early important design decisions and trade-offs. The software application is built onto it. The modifications in the software architecture later on in the development process cost dearly.

According to Kruchten, software architecture is used for [2]:

- Understanding what the system does and how the system works
- Thinking and working in the pieces of the system
- Extending the system
- Reusing the parts of the system to build other ones

Thus, a software system architecture helps us to answer most important questions related to a product. It also helps us to achieve high quality software.

In this paper, we present a software system architecture with its development process. The development process is explained via a weapon system example, a mine neutralization system for mine warfare ships. Weapon systems are complex and safety-critical embedded systems in general [3]. Analysis and design of these systems pose many challenges [4]. Most of those challenges can be addressed with a well-crafted software architecture. Such challenges and strategies to resolve them are presented with the associated architectural solutions throughout the development of the mine neutralization system example.

A new architectural style, named star-controller, is introduced, as well. An architectural style describes the structure of a pattern that can be applied to a family of systems. Architectural styles also explain the terminology of the components and connections along with a set of rules on how they can be combined [5]. How the style is applied to the architecture development is also provided.

The rest of the paper is organized as follows. Section 2 presents a brief discussion of how nonfunctional requirements are met through software architecture. Section 3 introduces the system and presents the architectural development process along with the system architecture and design diagrams. Section 4 draws the conclusion. Experiences, lessons learned and future work is explained in section 5.

2. Nonfunctional requirements through software architecture

Requirements engineering process provides the main input for the software architecture development. The software architect takes the requirements and develops a software architecture that meets both the functional and nonfunctional requirements. The

decisions he makes at this phase of software development establish the boundaries of the system quality attributes such as extensibility, modifiability, adaptability, reliability, safety, maintainability, testability etc. The importance of meeting quality attributes with software architecture is already recognized. For example, Bachmann and Bass present an attribute driven design method for designing the software architecture [6]. Bass and John link the usability to software architecture patterns [7].

In our weapon system example, adaptability, modifiability, maintainability, usability, testability, reliability and safety are the quality attributes that are specifically addressed. These attributes and how to achieve them are presented with specific architectural patterns and solutions. Analysis of the architecture of a software system reveals whether the architecture of that system is capable of meeting the system nonfunctional requirements.

3. Mine Neutralization System (MNS)

Due to developments in electronics and software systems, navies around the world undergo major revisions in their combatant ships. Instead of designing and building ships from the scratch, it is cheaper to revise its combat systems to increase the ship's combat capabilities. The Mine Neutralization System (MNS) is conceptualized and designed to adapt latest technologies in mine warfare without undergoing major changes in a mine hunting ship's original structure. The objective of the MNS is to detect and eliminate sea mines. The system uses a detection sonar to detect an underwater threat, possibly a sea mine. Classification of sonar data input helps the sonar operator to classify the threat type which may be a magnetic or moored mine. The operator of the system eliminates the mines via a remotely operated underwater vehicle (ROV). MNS encapsulates and controls all these main and auxiliary devices including system consoles to achieve sea mine hunting mission for navy mine warfare ships. Figure 1 depicts the use of the system in a mine hunting ship.

3.1. MNS High-Level and User-Level Goals

Requirements engineering is an important success factor in software projects [8]. The high-level and user-level goals of the system were identified through a series of interviews with navy officers who are the major stakeholders for this system. Also, the analysis of business opportunities and technological improvement projections for mine warfare systems guided the most important system requirements.

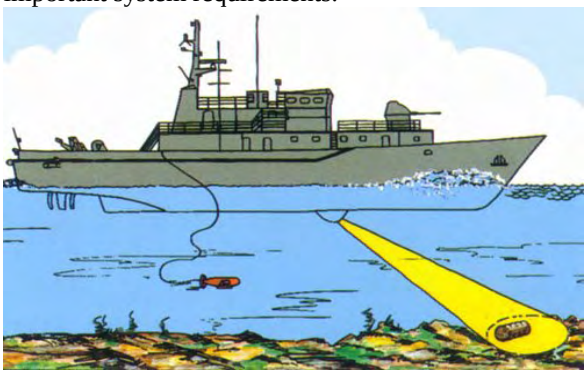


Figure 1. The illustration of the MNS use on a mine hunting ship

The interviews with navy officers revealed important shortcomings of existing mine hunting systems. For example, existing systems require quite a few personnel. In a mine hunting ship, the number of personnel is limited and sometimes operators need to stay on watch for long hours, which poses a threat to the mission. MNS reduces the number of personnel to only one operator. This is one of the important achievements of the system. Another accomplishment of the system is that the system is highly adaptable to the new technological advances in mine warfare. This requirement is derived from the business opportunities.

After a detailed requirement analysis, the high-level goals of the system are identified. Some of them are listed as follows:

- MNS provides a complete solution to satisfy the prospective technological advances in mine hunting warfare.
- MNS is a reliable and safe system that can eliminate the shortcomings of current systems in navies.
- The system is maintainable that is a benefit to both the developer and the customer.
- The system operates in 15-600 ft. depth range which is highly sufficient for the mine hunting operations.
- With the support of the umbilical cable attached to the mine neutralization vehicle (MNV), the length of the mission will not be limited to short periods.
- The system is highly adaptable for future upgrades.
- Emergency mode operation provides flexibility during mission.
- The system can operate with many existing sonar suites currently used in mine warfare operations.
- A large variety of alarms help the system operator to monitor the safety of the system.

The user-level goals are as follows:

- MNS requires only one operator.
- The system needs less training than existing systems.

- MNS has a simple graphical user interface which helps the operator and the commander of the ship to visualize the controls and the state of the system.
- The camera on the MNV provides high reliability for the operation.
- The easy control of the MNV provides an increase in the flexibility of the operation.
- Multilanguage support makes the product usable by many different countries without further training in language.
- Logging features of MNS helps the ship's crew prepare after-operation review reports.
- Logging features helps the personnel for maintenance of the system.
- Training mode is the same with the operation mode, which provides an excellent training environment for the ship's crew.

The requirements analysis phase of the system development lead to the following outstanding features of the proposed product:
- One-man operated system
- Highly adaptable to new sonar systems and remotely operated underwater vehicle systems
- A complete solution
- A simple and well-designed interface
- Easy training
- Long-operation support
- Emergency operation mode
- A safe and reliable system
- Multilanguage user interface
- A large variety of alarms and monitoring features
- Enhanced logging of conditions and operation milestones

All these goals and resulting features provide the most important input for the system software architecture development.

3.2. MNS Components

Analysis of similar mine warfare systems, reveals the necessary main components for the MNS. The system is composed of five main components:

- Detection and Classification Sonar Suite: The sonar suite is responsible for the detection and classification of mines. Two different sonars exist in this suite. The detection sonar is a long-range wide-spectrum sonar that is used to detect the presence of an underwater object. The classification sonar is used for further analysis of underwater objects suspected to be a mine threat. This sonar creates a contact for the system. Bat thermograph and echo sounder are the auxiliary devices attached to the suite to provide necessary sea condition data.

- Navigation Unit: This unit provides the precise location information of the ship. The navigation unit consists of a global positioning system (GPS) device and a gyro unit. Both of these devices provide the location data and one of the devices is sufficient for the operation. Therefore, the failure of one of the devices doesn't compromise the mission.

- Mine Neutralization System Console: This unit is the interface between the operator and the system.

- Mine Neutralization Vehicle (MNV): This unit handles the elimination of the sea mines. It is a remotely operated underwater vehicle and attached to the mother ship with an umbilical cable that carries the communication and power cables. The vehicle carries many devices to achieve the mission. Depth unit, TV camera, light, emergency pinger, umbilical cable, gyro unit and mine neutralization vehicle control unit are only some of them.

- Mine Neutralization System Controller: This unit is the heart of the system. It provides communication between components. It also synchronizes the events within the mine hunting operation. The mine system controller encapsulates the necessary interfaces in case the decision of using existing components in various types of mine hunting ships.

Figure 2 shows the connections between the components and the framework for the system software architecture.

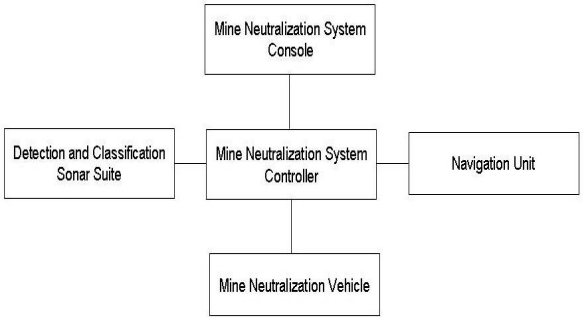


Figure 2. The MNS framework

3.3. MNS Software Architecture

Weapon system software development is an expensive and effort-intensive process and these types of systems tend to have long life-cycles. The systems evolve during the years. Older versions are replaced with newer versions in order to keep up with advancing technology. A well-designed software system architecture prolongs the system life-cycle and ease the maintenance effort.

Hofmeister et. al. describes the four views of the software architecture[1]. These four views are conceptual, module, code and execution view. The conceptual view deals with the issues relating to the application domain. One of the most important questions answered with the conceptual view is how the system fulfills its requirements. How the functionality partitioned to the conceptual components is also explained with the conceptual view. The module view explains how the conceptual components are mapped to subsystems and modules. In this view, the conceptual solution is realized with today's software platforms and technologies. The execution view describes the runtime interactions of the software application. It also deals with how subsystems and modules are mapped to the hardware platforms. The code view deals with how runtime entities are mapped to the deployment components such as executables, libraries etc. Each view acts an input for another view and helps the software architect to analyze trade-offs.

Developing the views of the system software architecture starts with a global analysis.

3.3.1. Global Analysis. The global analysis is the process of identification of factors that influences architectural design. The goal of the process is to develop strategies for each identified factor. The factors related to the development of MNS are listed as follows:

1. The quality of the product is more important than the schedule.
2. The system must be easily modifiable.
3. Because MNS is a weapon system, safety and reliability is extremely important.
4. The system must have a friendly user interface that minimizes operator errors.
5. MNS must be an adaptable system and incorporating COTS products must be easy.
6. The system is intended for many countries. Therefore, the user interface should easily be adaptable for different languages.
7. The system must be a maintainable system.
8. A one-man operated system is a must.
9. MNS should meet performance criteria.
10. The system design should support a 20-25 year life cycle.

During MNS development, strategies are laid out to provide solutions for each identified factor. It is important to cover each of the factors with at least one strategy. Table 1 shows factors and corresponding strategies. For example, factor 2 enforces the system to be easily modifiable. Encapsulating the features into separate components is selected as a strategy to ensure a solution for the specific factor. In the MNS framework, the navigation unit handles all the navigation tasks and the unit is only connected to the system controller. If an upgrade becomes necessary in the navigation features, the navigation unit can easily be replaced with a newer version. Such modification doesn't affect other components of the system. This is also how one of the corresponding high-level user goals is achieved.

The next step in the process is the development of the conceptual view of the system. The framework of the system presented in figure 2 is used as an input for the development of the conceptual view.

Factors	**Corresponding Strategy**
1,2,4,7,9,10	Use of well-known patterns
1,3,5,7,9,10	Build instead of buy and/or build products similar to the ones in the market
2,4,5,6,7,10	Make it easy to add and remove features
2,3,5,7,8,9,10	Use a central controller component
1,2,3,4,5,6,7,10	Use standards
2,4,5,6,7,8,9	Separate components and modules along dimensions of concerns
4,6,8	Decouple the user interaction module
2,7,9,10	Encapsulate features into separate components

Table 1. Factors and corresponding strategies

3.3.2. Conceptual view. In the global analysis, we decided to use the well-known patterns as a strategy to address some of the identified factors. For our product's conceptual view, we determined to use the Model-View-Controller pattern. The suggested context for this architectural pattern is interactive applications with a flexible human-computer interface [9]. The model-view-controller architectural pattern divides an interactive application into three components. The model contains the core functionality and data. Views display the information to the user. The views and controller together compromise the user interface. A change-propagation mechanism through controller ensures consistency between the user and the model. Figure 3 shows the conceptual view and how the

architectural pattern is applied to the MNS framework. The rationales for selecting the pattern are as follows:

- The product market is intended for the Navies around the world. This requires complying with the existing user interfaces from all over the world forcing the user interface of the application to be flexible. Like multiple language support, different look and views.
- Even if the model changes, the users will require the same information from the system. So the pattern enables us to decouple the model from the view. For example, an upgrade in the navigation unit will not affect the interface. The navigation unit is one of the components of the model and the mine neutralization system console, which is the interface to the user, is the view in the pattern.
- The product is a weapon system and therefore, it is a real-time interactive application.
- Abstracting the controller enables us to focus on the synchronization of the events in the system in a real-time environment.
- The system is an adaptable system which requires modification when necessary in each component of the pattern.
- Easy addition of views and controllers will benefit the maintenance of the product.
- The architecture of the product will base a framework for future versions and similar products. It is important to remember the necessity of the long life-cycle of the product.

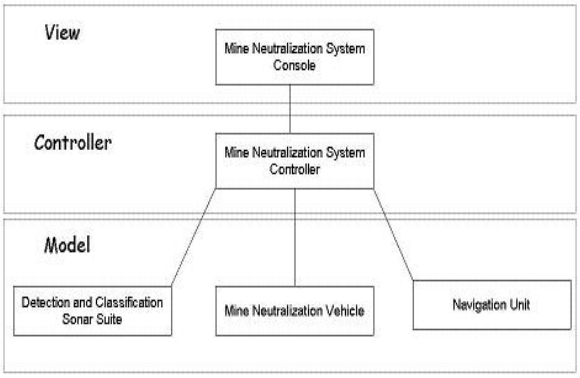


Figure 3. Conceptual view

Selection of the model-view-controller pattern helped us to conceptualize an adaptable and maintainable system. This is how important nonfunctional requirements can be achieved in the conceptual view.

3.3.3. Module view. The main purpose of the module view is to simplify the system's implementation in software. It helps us to overcome the complexity of the system. In the module view, all the application functionality, control functionality, and meditation are mapped to subsystems, modules and connections.

For the module view, we developed an architectural style named star-controller architecture. The style resembles to a star network topology in structure. The style benefits from the well-known design decomposition principle. The system is carefully partitioned to subsystems which are strictly loosely coupled with each other. In this architectural style, the system is divided into two types of components: controllers and subcomponents. The controllers handle the control functionality and the subcomponents handle the application functionality in the module view. The architectural style follows two basic rules:

1. A controller can be connected to controllers and subcomponents.
2. Subcomponents can only be connected to controllers.

Figure 4 shows the star-controller architectural style.

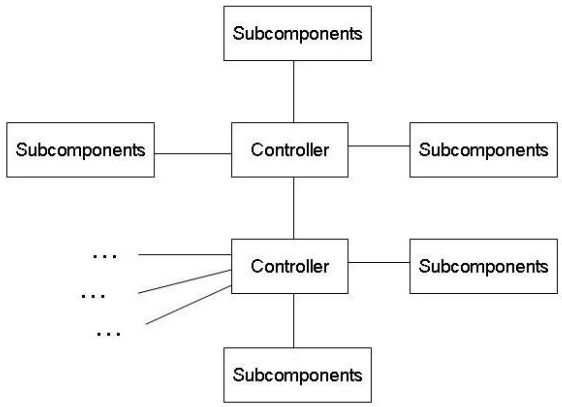


Figure 4. The star-controller architectural style

The style helps us to reduce the development effort for interfaces and similar subcomponents. It also enables the independent development of subcomponents or easy addition of existing subsystems which is enforced to MNS with one of the high-level goals. This architecture ensures to achieve an adaptable and maintainable system.

In this architectural style, faults can easily be identified and localized to some specific portion of the system. Subsystems are tested separately and integration testing is achieved as new subsystems are added to the system. The style follows design for testing principle in this perspective.

The star-controller architecture has a simple structure. Synchronization and the control flow of

information are handled by controllers. The information is produced by subcomponents. The controllers' solemn task is to ensure reliable communication and synchronization which are important considerations for real-time systems. In this architecture, high cohesion is achieved via attributing a part of functionality per controller.

The nonfunctional requirements of MNS require the system to be safe and reliable. Ease of testing and a simple design is essential for achieving safety and reliability.

The major drawback of the style is that the failure of one controller disables all the subcomponents attached to it. In the MNS example, the solution is provided with the redundancy in hardware. The MNS has a system self-checking mechanism built in its design. Every controller constantly monitors the attached subcomponents and another controller. Whenever a failure is detected in a subcomponent or in a controller, the system immediately switches to the redundant hardware. Only some of the key elements have this redundancy. Another solution to this problem may be redundancy through software. A software module having the same functionality may be designed differently and installed to the redundant hardware. However, only hardware redundancy exists in the MNS. Figure 5 shows how star-controller architecture is applied to the mine neutralization vehicle subsystem. Note the self-checking mechanism is accomplished via status attributes in classes.

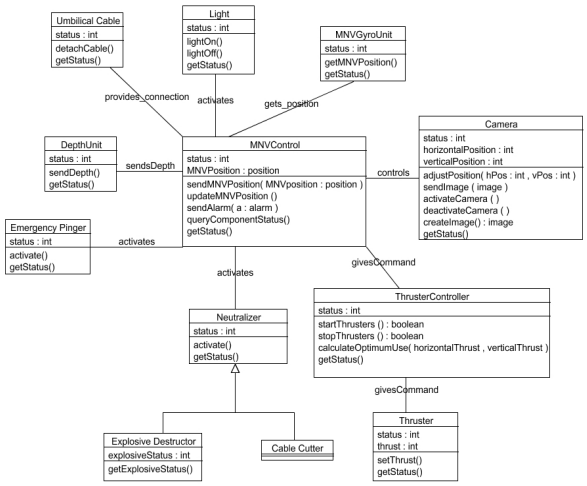


Figure 5. The mine neutralization vehicle subsystem

Rationales on choosing the star-controller architectural style are listed as follows:

- Easy elimination of synchronization problems will increase the system's reliability and safety.
- Functions and controls are separated. Therefore, modifications in functionality will not affect the control aspects.
- Easy addition/removal of modules and functionality, thus support for adaptation and modification.
- Easy localization of errors will reduce the testing effort.
- Elimination of errors and fault propagation increases the system safety and reliability.

A system may have orthogonal software architectures that address different concerns. Because high reliability and safety are important concerns for MNS, we used an additional software architecture addressing communication and synchronization issues. A layered architectural pattern is selected. Layered architectures help us to structure applications that can be decomposed into groups of subtasks. These subtasks are at a particular abstraction. In the MNS example, the system is divided into two layers. The first layer, networking layer, handles the communication between modules as well as establishing the protocols and checking messages for errors. The networking layer corresponds to the physical and data link layer in open system interconnection model (OSI). The second layer is named system layer and it is responsible for all other application-related communications in the system. Figure 6 shows the layered architecture of the MNS.

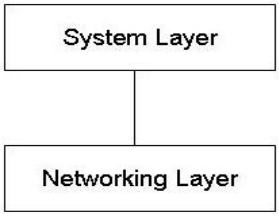


Figure 6. The layered architecture of the MNS

The next phase in the process is the process is the development of code and execution views of the system. However, these are detailed design views and are will not be addressed in here.

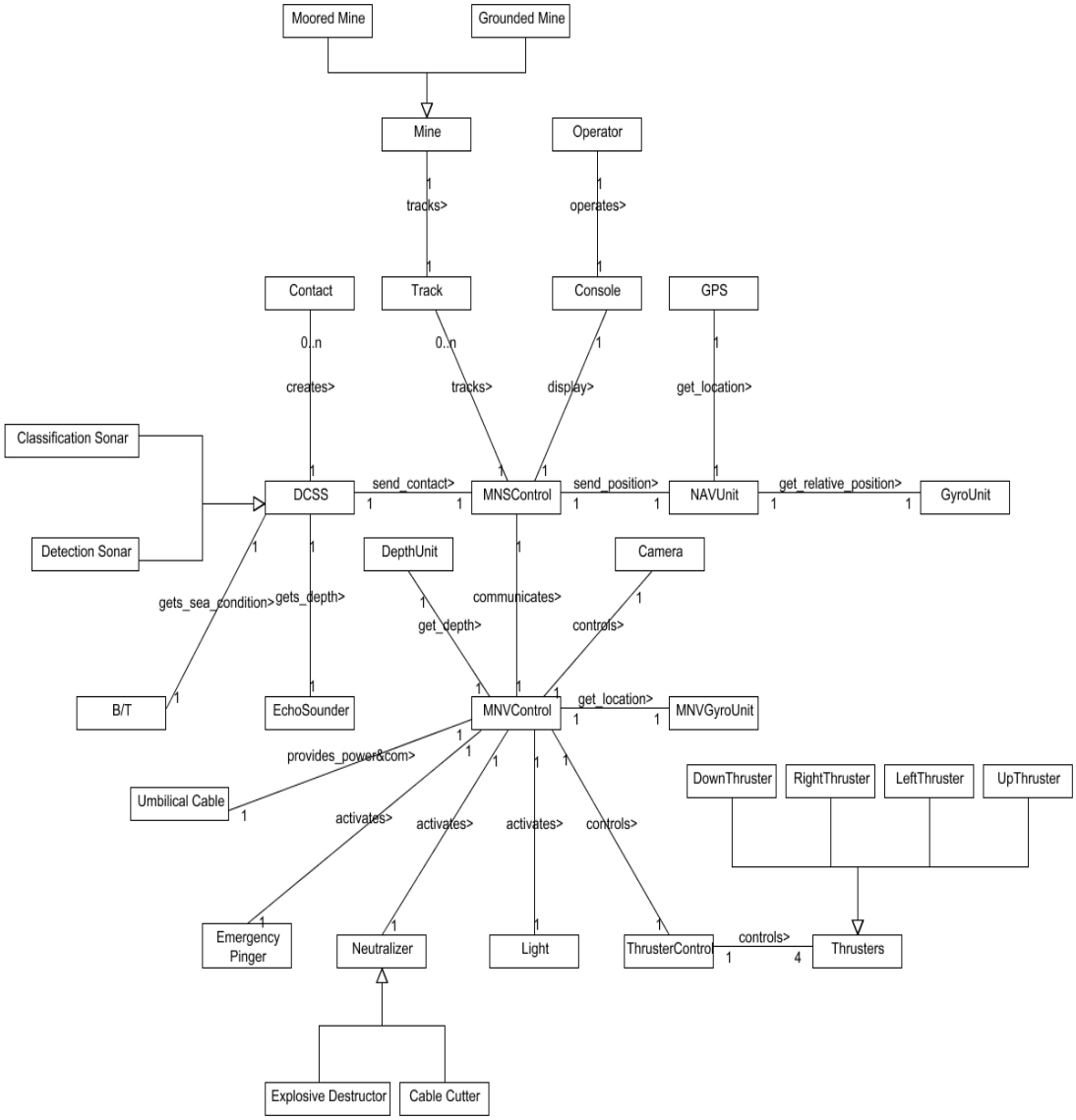


Figure 7. The domain model of the MNS

3.4. MNS High-Level Software Design

The inputs from different views of MNS architecture are used in the high-level design of the system. It is imperative that the design follows the architectural design decisions. A smooth transition from one activity to another activity is achieved in the MNS example using the architectural decisions and rationales. Figure 7 shows the derived domain model of the MNS. Note the structural similarity of the domain model and the star-controller architectural style introduced earlier. Figure 8 and 9 show the respective high-level design examples from the system.

4. Conclusions

In this paper, we presented the development of a real-world weapon system software architecture example. Weapon systems development is a long and expensive process. These types of systems are generally complex safety-critical embedded systems.

Because of these properties, high quality is imperative to accomplish in weapon systems. Only, a well-designed architecture can achieve all the necessary nonfunctional requirements.

First, we identified the high-level and user-level goals through interviews with navy officers and analysis of similar existing systems. Also, analysis of

similar systems revealed the necessary components for the mine neutralization system. Then, the global analysis helped us to identify factors that influence our architectural design decisions. The strategies to resolve the factors are determined. The global analysis guided the development of the conceptual and module views of the system software architecture. The commonly-known patterns applicable to the product are used and a new architectural style is developed to meet the specific properties imposed by the previously identified factors. Finally, we showed how the architecture formed as an input for the high-level system design.

We introduced the star-controller architectural style within the system architecture development. This style has the advantage of

- Being simple and easily testable
- Achieving low-coupling and high-cohesion
- Having increased control over synchronization and communication needed in real-time systems.

The major drawback of the style is that the failure of a controller also disables the subsystems attached to it. However, in our context this drawback is not an issue. In weapon systems we require a fully functioning system. Impaired system functionality is not acceptable. Hardware redundancies are used overcome this drawback.

In the example, we used model-view-controller architecture pattern and star-controller architecture to achieve usability, extensibility, adaptability, modifiability, testability, maintainability, safety and reliability. The layered architecture is used to increase maintainability, safety and reliability.

5. Experiences, Lessons Learned and Future Work

During the system architecture development, we understood that we won't able to satisfy all the nonfunctional requirements with one particular architecture pattern. The requirements we had forced us to use multiple software architectures for different nonfunctional requirement sets. This was a major finding and experience we had during development. Some of the lessons learned can be listed as follows:

- Developing software architecture is an organized and planned activity which takes the nonfunctional requirements into consideration as well as the functional ones.
- Paying special attention to the requirements gathering phase is a good promise of a successful software architecture development. The views of the navy officers about the system proved to be very critical at this phase.
- Partitioning the task into different architectural views, each addressing separate concerns, proves to be very useful in meeting both functional and nonfunctional requirements and reducing the cost of software development process.
- A well documented conceptual view ensures that the problem at hand is understood by all the stakeholders. Communication among developers is improved this way and misunderstandings between customers and engineers are reduced if not eliminated completely.
- Conceptual components ease the way we create module view which identifies static structures and layers of the system being developed. Conceptual view also establishes a starting point for the identification of a simple execution view.

Future work may include:

- One of the challenges we had was the necessity of incorporating redundancy. We eluded software redundancy by using hardware redundancy. However, we would like to see how software redundancy interacts with software architectures.
- Researching architecture description languages (ADLs) have been the focus of software architecture community in the recent years [10]. It is possible to analyze software architectures with ADLs. We would like to analyze the star-controller architectural style with ADLs and get an in depth understanding of the style.
- We would also like to see how the proposed style would be beneficial in other systems.

6. References

[1] Hofmeister, C., Nord, R., and Soni, D., *Applied Software Architecture*, Addison-Wesley Object Technology Series, New Jersey, 2000.

[2] Kruchten, P., *The Rational Unified Process: An Introduction*, Addison-Wesley, Reading, MA, 1999

[3] Demir, K.A., *Analysis of TLCharts for Weapons Systems Software Development*, Master's Thesis, Naval Postgraduate School, Monterey, CA, USA, December 2005

[4] Drusinsky, D., Shing M., Demir, K. "Test-Time, Run-Time, Simulation-Time Temporal Assertions in RSP, *Proceedings of 16th International Workshop on Rapid System Prototyping, (RSP'05)*, Montreal, Canada, June 2005, pp. 105-110

[5] Garlan, D., and Shaw, M., "An Introduction to Software Architecture", *Advances in Software Engineering and*

Knowledge Engineering, World Scientific Publishing Componay, December 1993, pp. 1-39

[6] Bachmann, F., and Bass, L., "Designing Software Architecture for Quality: the ADD Method", OOPSLA, Tampa Bay, Florida, USA, October 2001

[7] Bass, L., and John, B. E., "Linking usability to software architecture patterns through general scenarios", *The Journal of Systems and Software*, Vol. 66, 2003, pp. 187-197

[8] H.F. Hofmann, and F. Lehner, "Requirements Engineering as a Success Factor in Software Projects", *IEEE Software*, July/August 2001, pp. 58-66.

[9] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M., *A System of Patterns*, John Wiley & Sons ,West Sussex, England, 1996.

[10] Medvidovic, N., and Taylor, R. N., "A Classification and Comparison Framework for Software Architecture Description Languages." In *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp.70--93, 2000.

Acknowledgements and Disclaimers

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any affiliated organization or government.

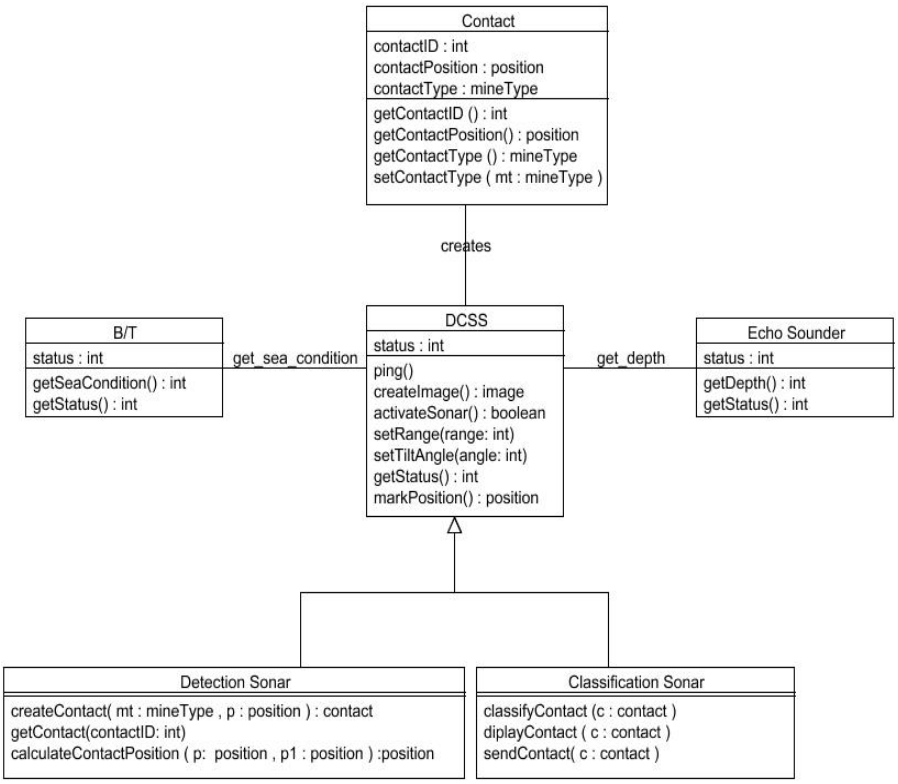


Figure 8. The detection and classification sonar suite high-level design

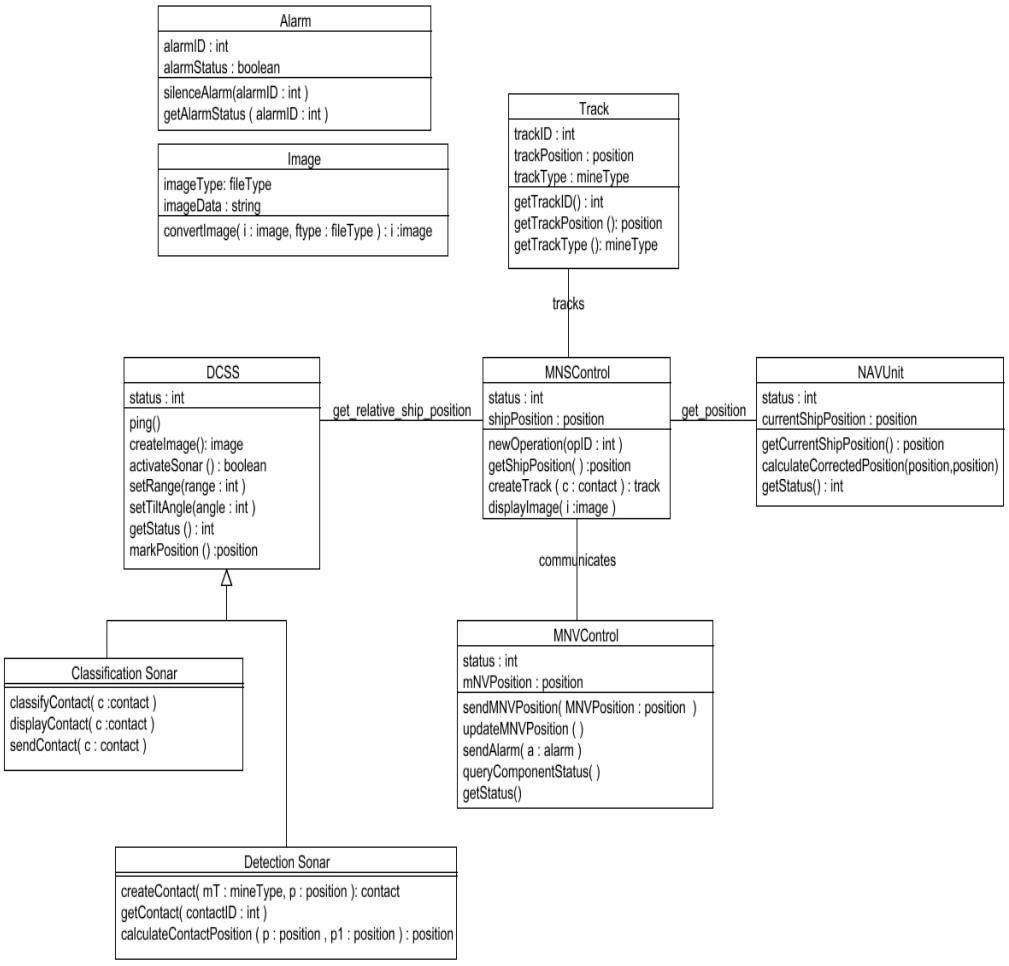


Figure 9. The mine neutralization system controller high-level design

Model-Güdümlü Mimari

Model Güdümlü Yazılım Mimarisine Dayalı

Elektronik Harp Benzetim Senaryosu

Hazırlama Yazılımı Geliştirimi

Ersin ÜNSAL, Alp Bülent Burç SÜRMELİ
HAVELSAN AŞ Eskişehir Yolu 7. km 06520 Çankaya / ANKARA
{eunsal, asurmeli}@havelsan.com.tr

Özet

Bu çalışmada, askeri bir proje kapsamında geliştirilen Elektronik Harp Benzetim Senaryosu Hazırlama Yazılımı'nın geliştirilme süreci anlatılmaktadır. Bu yazılımın geliştirilmesi aşamasında tanımlı teknik isterler, proje kaynakları kullanım koşulları ve sahip olunan teknolojik imkan ve kabiliyetler dikkate alınarak yapılan değerlendirme sonucunda Model Güdümlü Yazılım Mimarisi yaklaşımının kullanılmasının hem son derece emek-etkin bir seçenek olduğu hem de kurumsal kültüre olumlu katkı sağlayacağı değerlendirilmiştir. Kurumsal kültürün Model Güdümlü Yazılım Mimarilerine dayalı kayda değer geçmiş deneyiminin olmaması ile birlikte gerek bahse konu yaklaşımın, gerekse bu yaklaşımı destekleyen araçların da hali hazırda yeterli yaygınlığa, olgunluğa ve standarda ulaşmamış olması sebebiyle ticari bir araç seti tedarik edilmesi yerine serbest kullanım lisansı olan alternatiflerin incelenmesi değerlendirilmiştir. Proje zaman kısıtları sebebiyle kısa süren inceleme süreci sonucunda Eclipse aracının ve bu aracın bir parçası olarak çalıştırılabilen Model Güdümlü Yazılım Mimarilerini destekleyici eklentiler (plug-in) setinin kullanılması kararı alınmıştır. Elektronik Harp Benzetim Senaryosu Hazırlama Yazılımı, Eclipse platformu üzerinde, Eclipse Modeling Framework ve Eclipse Rich Client Platform teknolojileri kullanılarak geliştirilmiştir. Eclipse Modeling Framework'un sunduğu otomatik modelleme ve kod üretme mekanizması ve Eclipse Rich Client Platform'un sağladığı platform bağımsız görsel arayüz geliştirme imkanları kullanılmıştır. Bu çalışmada, ayrıca OpenArchitectureWare isimli Model Güdümlü Yazılım Mimarisi uyumlu bir çerçeveden de faydalanılmıştır. Bahsedilen yöntem ve araçlar kullanılarak, oldukça kısa bir sürede senaryo hazırlama yazılımı başarıyla geliştirilmiştir. Modelde oluşan değişimler kolayca yazılıma yansıtılabilmiş, modelden otomatik olarak üretilen yazılım kodu test aşamalarından başarıyla geçmiştir. Bu bildiri, yapılan çalışmalar kapsamında elde edilen bilgi ve tecrübeyi paylaşmak amacıyla kaleme alınmıştır.

1. Giriş

Elektronik Harp Benzetim Senaryosu Hazırlama Yazılımı mimari tasarımını doğrudan etkileyeceği düşünülen ana teknik isterler şu şekilde listelenebilir:

a. Yazılımın girdi ve çıktı veri yapılarının (senaryo ve destekleyici veri yapıları) çok amaçlı kullanımı,
b. Geliştirilecek yazılımın girdi ve çıktı veri yapılarına sıkı sıkıya bağlı olması,
c. Çok amaçlı kullanımları sebebiyle girdi ve çıktı yapılarının yazılımın geliştirim süreci boyunca yüksek oynaklığa sahip olacağı ve buna bağlı olarak gerçekleşmesi muhtemel yeniden işleme faaliyetlerinin çok olması beklentileri,
d. Yazılımın kullanacağı girdilerin ve üreteceği çıktıların tam ve eksiksiz olarak yazılımın gerçekleştiriminden önce tanımlı ve kullanıma hazır halde oluşu,
e. Yazılım alt yapısının istenilen kabiliyetleri ekleyip istenilen kabiliyetleri çıkarmaya imkan verecek şekilde eklenti geliştirilmesine imkan sağlaması.

Proje kaynakları kullanım koşullarının yazılım takımımızın önüne koyduğu riskler ise proje takvimi

ve yetişmiş, yeterli seviyede alan uzmanlığına sahip, tanımlı ve uyulması gerekli süreçler dahilinde geliştirme faaliyetlerine hızla katkı verebilecek insan gücünün istendiği anda temin edilememesinden oluşmaktadır. Bu koşulların sebep olabileceği gecikmeler ve hata oranı yüksek ürün geliştirilmesi gibi olumsuz sonuçlardan kaçınılabilmesi için insan hatalarını en aza indirecek bir yazılım mimari yaklaşımına ihtiyaç duyulmaktadır.

Teknolojik imkan ve kabiliyetler açısından ise yazılım tasarımı ve belli bir seviyede de yazılım mimari tasarımı yapmaya imkan verecek UML tabanlı teknolojileri destekleyen tasarım araçlarına ve bunları sundukları imkanlar dahilinde de yetkin olarak kullanabilecek bilgi birikimine sahip olunmakla birlikte, karşılanması hedeflenen teknik isterlerin mevcut araçlar kullanılarak belirlenen sürede karşılanamayacağı değerlendirilmiştir.

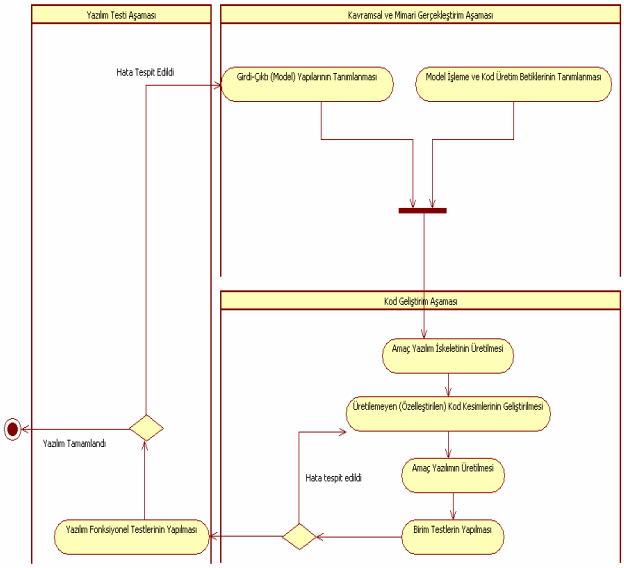
Sınırları çizilmiş ihtiyaç ve zorunluluklar Eclipse aracının ve beraberinde kullanılabilen Model Güdümlü Yazılım Mimarisi (MGYM) [1,2] yaklaşımını destekleyici nitelikteki eklenti uygulamalarının kullanılması kararının verilmesinde etkili olmuştur.

Eclipse [3,4], şirket-bağımsız uygulama geliştirme platformu ve yazılım çerçeveleri geliştirme amaçlı projeler üreten bir açık kaynak topluluğudur. 2001 yılında yazılım sektörünün öncü firmaları tarafından kurulmuş ve çok kısa bir sürede oldukça önemli ürünler ortaya çıkarmayı başarabilmiştir. Eclipse Tools Project, Eclipse Modeling Framework (EMF), Eclipse Web Tools Project (WTP), Eclipse Test & Performance Tool Project (TPTP) ve Eclipse Business Intelligence and Reporting Tools (BIRT) projeleri popüler Eclipse ürünleri arasındadır.

Geliştirilmekte olan Elektronik Harp Benzetim Senaryosu Hazırlama Yazılımı, Eclipse Yazılım Geliştirme Platformu üzerinde, Java programlama dili kullanılarak geliştirilmektedir. MGYM yaklaşımı benimsenmiş olup, bu yaklaşımı destekleyen Eclipse Modeling Framework ürünü kullanılmaktadır. Elektronik Harp Benzetim Senaryosu Hazırlama Yazılımı'nın görsel arayüzü ise Eclipse Rich Client Platform ürünü kullanılarak geliştirilmektedir. Model dönüşümü ve model birleştirme işlemleri için OpenArchitectureWare'den [5] faydalanılmaktadır. Bahsi geçen ürünlerin tamamı açık kaynaklı ürünler olup, ücretsiz olarak dağıtılmaktadırlar.

2. Model Güdümlü Yazılım Mimarisi

Model Güdümlü Yazılım Mimarisi (Model Driven Architecture - MDA), OMG (Object Management Group) tarafından önerilen bir yazılım geliştirme mimarisidir. Bu mimariye göre öncelikle yüksek seviyeli, platform bağımsız bir uygulama modeli hazırlanır. Bu modele PIM (Platform Independent Model) adı verilir ve bu modeling amacı altyapıdan bağımsız bir şekilde uygulamayı tasarlamaktır. Daha sonra dönüşüm teknolojileri kullanılarak, PSM (Platform Specific Model) adı verilen platform uyumlu bir model üretilir. Son olarak da PSM üzerinden yazılım kodu üretilir. Model Güdümlü Yazılım Mimarisi UML, MOF ve XMI gibi bir çok modelleme standardını desteklemektedir. Şu an için bir model üzerinden karmaşık bir sistemin kodunun tamamen üretilemeyeceği çok açık olsa da, MDA sayesinde tekrar tekrar yazılması gereken bir çok yazılım parçası otomatik olarak üretilebilmektedir. Şekil 1'de model güdümlü yazılım geliştirme süreci gösterilmektedir.



Şekil 1. Model Güdümlü Yazılım Geliştirme Süreci

MGYM'nin en önemli gerekleri şu şekilde sıralanabilir.

a. Model ve meta-model tanımlama, sorgulama (ve analiz) ve dönüştürme dillerinin varlığı,
b. Model Tanımlama arayüzlerinin varlığı,
c. Meta-modeller kullanılarak etkinlik alanına özgü (Domain Specific) varlık, ilişki ve kısıtların tanımlanabilmesi,

d. Etkinlik alanındaki bir tasarım kararını görselleştirmeye ve ilgili modeli üretmeye imkan verecek, etkinlik alanına özel diyagram(lar) üretilebilmesi,
e. Ortaya konan modelden, amaç kod üretilmesine kadar yaşam döngüsüne destek verebilecek katmanlı ve yapısal kod üretilebilmesi.

Eclipse platformunda değişik proje gruplarınca geliştirimi devam eden MGYM'ye dayalı teknoloji eklentileri hali hazırda kullanıcıların kullanımına sunulmuş bulunmaktadır. Yukarıdaki listede sıralanmış olan kabiliyetlerin temini amacıyla şu eklentiler geliştirilmektedir.

1. a maddesinde anılan gereği sağlayabilmek amacıyla ATL (Atlas Transformation Language), Viatra ve OpenArchitectureWare eklentileri geliştirilmekte ve burada bahsi geçen ihtiyacı karşılamak üzere Model Bus adı verilen ve Eclipse içinde belli bir model üzerinde bir işlem gerçekleştirecek tüm uygulamaların, sağlanan ortak arayüz ile erişebilmelerine imkan verecek bir çeşit model protokolü geliştirilmektedir.
2. b maddesinde anılan gereği sağlayabilmek üzere de Eclipse içinde gerek "ecore" ve "KM3" gibi meta-model tanimlama dillerine ve bu dillerden gerek birbirlerine gerekse XML, XSD gibi veri tanımlama dillerine çevirimler için gerekli alt yapı Eclipse Modelling Framework (EMF) ve OpenArchitectureWare grupları tarafından sağlanmaktadır.
3. c maddesinde bahsi geçen etkinlik alanına özel kavramlar ve bunlar arasındaki ilişkiler "ecore" ve "km3" gibi meta-model tanımlama dilleri ile sağlanmaktadır. Bu meta-model dilleri kullanılarak geliştirilen modeler için otomatik nesne kısıt bilgileri (OCL) üretilebilmektedir.
4. d maddesinde bahsedilen gereğin sağlanması içinde iki alternatif çözüm üretilmiştir. Bunlar biri, ötekine de alt yapı sağlayan ve EMF ile Graphical Editing Framework (GEF) arasında köprü kurma kabiliyetindeki Graphical Modelling Framework (GMF) eklentisi ile sağlanmaktadır. Benzer bir yapıyı sunan diğer bir eklenti ise Topcased [7] eklenti grubudur. Topcased alt yapı olarak GMF'i kullanırken GMF'deki yapısal kullanım zorluklarını aşmak üzere kimi kullanım kolaylıkları sağlayacak şekilde geliştirilmiştir. Topcased ayrıca tarif ettiği bu yöntemin hayata geçirildiği örneklere de dağıtımı içinde yer vermektedir.
5. e maddesinde anılan ihtiyacı karşılamak üzere de daha önceki maddelerde ismi geçen OpenArchitectureWare ve EMF eklentilerinin kullanılabildiği söylenebilir.

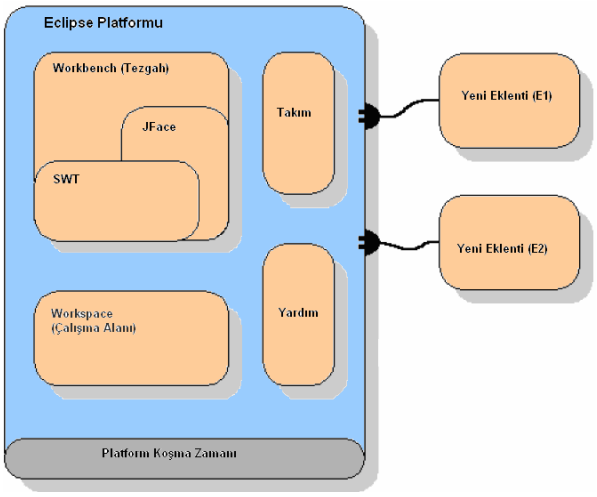
Eclipse'in sağladığı bu teknolojiler, henüz yeni yeni olgunlaşmaya başlamış olmakla birlikte mevcut kabiliyetleri ile MGYM'ye dayalı teknolojilerin geleceğini şekillendirmek üzere standartları belirlemeye aday ve muktedir uygulamalardır.

Geliştirilen Elektronik Harp Benzetim Senaryosu Hazırlama Yazılımı kapsamında EMF tabanlı teknolojilerden yararlanılmış ve ihtiyaç duyuldukça da OpenArchitectureWare teknolojisine başvurulmuştur.

3. Eclipse Mimarisi

Eclipse, Java programlama dili ile geliştirilmiş, açık kaynaklı, programlama dillerinden bağımsız olarak farklı uygulama geliştirme araçları için evrensel bir platform sunan bir projeler kümesidir.

Eclipse'in açık kaynaklı ve genişleyebilir mimarisi, temelde eklenti (plug-in) mantığına dayanmaktadır. Eklentiler, Eclipse üzerindeki en temel işlev birimleridir. Eclipse'e kazandırılmak istenen yeni işlevler, eklentiler aracılığıyla kazandırılır. Eklentiler, genişleme noktası (extension point) adı verilen bir mekanizmayı kullanarak yeni işlevler tanımlarlar. Yeni eklenen eklentiler, var olan eklentiler üzerindeki genişleme noktalarından faydalanacakları için, eklenti geliştiricilerin genişleme noktaları tanımlamalarına özen göstermeleri gerekmektedir.



Şekil 2. Eclipse Mimarisi

Şekil 2'de Eclipse Mimari görülmektedir. Platform Koşma Zamanı üzerinde görülen Eclipse Platformu, tüm eklentiler için ortak temel katmandır. Şekilde görülen JDT (Java Development Tools – Java Geliştirme Araçları), PDE (Plug-in Development Environment – Eklenti Geliştirme Ortamı) ve Workbench (Tezgah) Eclipse Platformu üzerine eklenebilen temel bazı eklentilerdir. E1 ve E2 ise uygulama geliştiriciler tarafından geliştirilebilecek eklentileri ifade etmektedir ve bu eklentiler kolaylıkla Eclipse Platformu üzerine eklenebilirler.

Şekil 2'de görülen SWT (Standart Widget Tool) katmanı, Eclipse'de kullanılan en temel görsel kütüphanedir. Eclipse geliştiriciler, Java'nın standart SWING kütüphanesi kullanmak yerine, SWT isimli kütüphaneyi tercih etmişlerdir. SWT kütüphanesinin en temel özelliği, işletim sisteminin görsel kütüphanelerini kullanması, dolayısıyla bir Java uygulamasından ziyade, işletim sistemi üzerinde geliştirilmiş yerel bir uygulama gibi çalışmasıdır. Bu şekilde işletim sistemlerinin görsel kütüphanelerinde bulunan çeşitli esnekliklerden ve özelliklerden faydalanılabilmektedir.

SWT kütüphanesi oldukça alt seviyeli bir kütüphane sayılabilir. Uygulama geliştiricilerin işini kolaylaştırmak amacıyla, SWT kütüphanesinin özelliklerini kullanan, JFace isimli daha üst seviyeli bir kütüphane geliştirilmiştir. JFace, SWT kütüphanesinin arayüzlerini saklamaz, onun üzerine kendi üst seviye arayüzlerini tanımlar. JFace, hızlı ve kolay bir şekilde zengin görsel arayüzler geliştirmeye olanak sağlar.

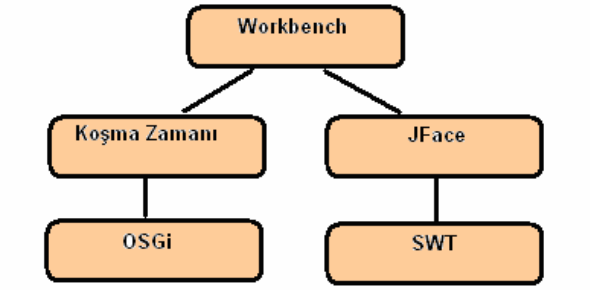
4. Eclipse Rich Client Platform

Rich Client Platform (RCP), aşağıdaki parçalardan oluşan bir yazılım parçası olarak tarif edilmektedir [6] :

- çekirdek katman,
- standart bir paketleyici çerçeve,
- taşınabilir görsel bir kütüphane,
- dosya arabellekleri, metin düzenleyicileri,
- workbench (tezgah),
- güncelleme yöneticisi.

RCP platformu kullanılarak daha hızlı uygulama geliştirme yapılabilir, geliştirilen uygulamalar farklı işletim sistemlerine kolayca taşınabilir ve geliştirilen uygulamalar geneneksel istemci kullanan uygulamalar kadar zengin öğeler içerebilir.

Eclipse RCP, yukarıda sayılan özellikleri sağlayan bir yazılım geliştirme platformudur. Eclipse RCP'nin yapı taşları Şekil 3'de görülmektedir.



Şekil 3. Eclipse RCP Bileşenleri

OSGi, Java için geliştirilmiş dinamik bir modül çerçevesidir. OSGi teknolojisi, küçük ve yeniden kullanılabilir yazılım parçalarının bir arada çalışabilmesi için gerekli yapı taşlarını içerir. Eclipse Koşma Zamanı, OSGi üzerine inşa edilmiş, ekleme noktaları ve eklentiler için gerekli desteği veren bir yapıdır.

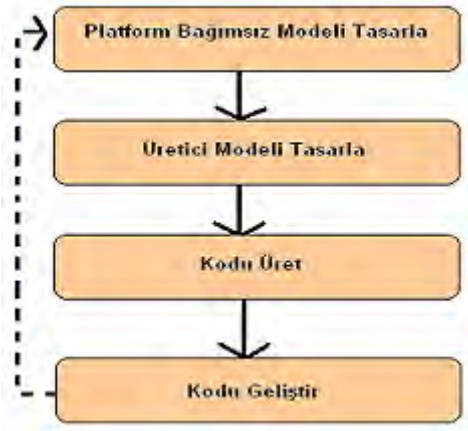
Workbench (Tezgah) ise, Koşma Zamanı, JFace ve SWT katmanları üzerine inşa edilmiş, görünümleri, editörleri, perspektifleri, eylemleri ve daha birçok şeyi yöneten, ölçeklenebilir görsel bir ortamdır.

RCP tanımındaki bileşenler Eclipse RCP için şu şekildedir. OSGi, paketleyici çerçeve görevi görürken, Eclipse Koşma Zamanı, çekirdek katman işlevini yerine getirir. SWT taşınabilir görsel kütüphane görevini yerine getirir. JFace ise metin düzenleyicilerini içerir.

5. Eclipse Modeling Framework (EMF)

EMF, MGYM uyumlu olarak geliştirilmiş açık kaynaklı bir yazılım çerçevesidir. EMF, hazırlanan bir

model üzerinden ilgili yazılım kodunu üretmeye olanak veren bir teknolojidir.

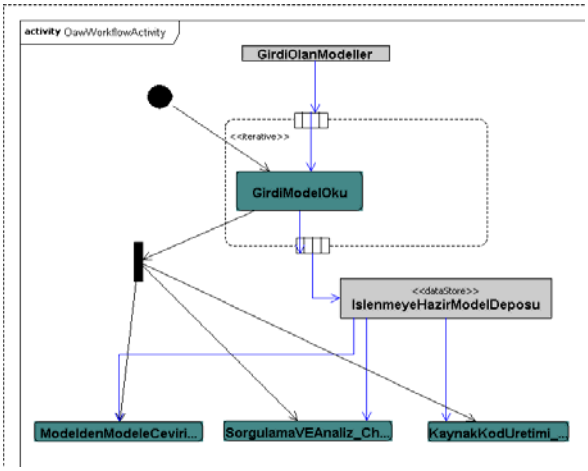


Şekil 4. EMF ile Yazılım Geliştirme Süreci

Şekil 4'de EMF kullanılarak yazılım geliştirme süreci gösterilmiştir. Şekilden de anlaşılacağı üzere, öncelikle platform bağımsız bir model (PIM) geliştirilmelidir. Daha sonra EMF aracılığıyla PIM model, genmodel adı da verilen üretici modele dönüştürülür. Üretici model üzerinden kod üretimi gerçekleştirilir. Daha sonra ise üretilen kod üzerinde geliştirmeler yapılabilir. Bu süreçlerin her birinden sonra çıktılar incelenip, gerekli görüldüğünde daha önceki aşamalara geri dönülebilir. Diğer bir ifadeyle, üretilen ve üzerinde geliştirme yapılan bir kod için modelde değişiklik yapılması gerektiğinde, sonradan geliştiriciler tarafından yapılan kod değişiklikleri bozulmayacak şekilde modeller ve kod tekrar üretilebilmektedir.

6. OpenArchitectureWare

OpenArchitectureWare (oAW), Java ile geliştirilmiş, MGYM uyumlu bir üretici çerçevedir. Modellerin ayrıştırılmasına, dönüştürülmesine ve kontrol edilmesine destek verdiği gibi modeller üzerinden kod üretimine de olanak sağlar. oAW, başta EMF olmak üzere XML, UML2 gibi birçok modelleme mimarisine ve çerçevesine destek vermektedir. Temelde kod üretimi ve model dönüşümü iş akışlarını tanımlamaya yarayan bir iş akışı motorundan oluşmaktadır. Model kontrolü için Check isimli bir sınama diline, model dönüşümü için Xtend isimli fonksiyonel bir model dönüşüm diline ve kod üretimi için Xpand2 isimli güçlü bir kod üretici dile sahiptir.



Şekil 5. Tipik bir oAW Akışı

Şekil 5'de tipik bir oAW kontrol ve veri akışı resmedilmeye çalışılmıştır.

Tipik bir oAW akışı öncelikle girdi olan model dosyalarinin iteratif olarak okunarak İşlenmeye Hazır Model Deposunda biriktirilmesiyle başlar. oAW'de herbir model bilgisinin yer aldığı yapıya slot adı verilmektedir. oAW'in bir parçası olarak geliştirilmiş olan girdi modellerini okuyan bileşen, gerek EMF tarafından tanımlanmış olan ecore tabanlı XMI dosyalarını gerekse üçüncü parti araçlar (Magicdraw, Poseidon, StarUML gibi) tarafından oluşturulan XMI dosyalarını okuyabilmektedir. Ancak göreli olarak EMF tabanlı XMI dosyaları üzerinde sağlanan imkan ve kabiliyetler daha gelişmiş ve olgunlaşmış durumdadır. Bir kez model dosyaları ilgili slotların içine yerleştirildikten sonra bu modeller üzerinde belli bir akış sıra düzeninde değişik işlemler yapılabilmektedir [5].

Bu işlemler bir modelden bir diğer modele çevirim, bir model içeriğinin değiştirilmesi, modellerin belli kurallar dahilinde birleştirilmesi ya da ayrıştırılması olabilmektedir. Böyle işlemleri yapmaya imkan sağlayan bileşen oAW'nin xTend bileşenidir. Bu bileşen yine aynı isimle anılan, tanım ve komut bloklarıyla tanımlı bir gramere sahip olan xTend dili ile tarif edilmiş işlemleri ve sorguları slotlar içine yüklenmiş modeller üzerinde çalıştırır. xTend dili gramer olarak xPand dili ile hemen hemen aynı özellikleri taşır. Aralarındaki en belirgin fark xTend dilindeki model elemanı yaratma (create) deyimidir. Bu deyim sayesinde xPand dili kullanılarak yapılamayacak olan bir modelden baska bir model üretme işlemleri gerçekleştirilebilmektedir [5].

Yine Şekil 5'de gösterilen ve oAW tarafından sağlanan diğer bir kabiliyet de Sınama (Check) deyimleri yazılarak model üzerinde yapılabilecek doğrulama işlemleridir. Sınama deyimleri, model

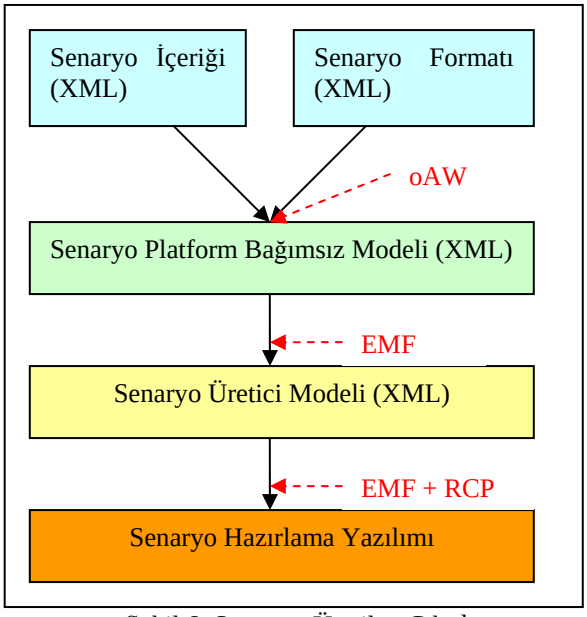
içindeki belli bir özelliğe sahip model elemanlarının seçilmesini ve yapılan seçimin belli kriterleri sağlayıp sağlamadığının sınanmasını sağlayarak model üzerinde istenen denetimlerin yapılabilmesini sağlamaktadır. Yaygın kullanım şekli belli bir model üzerinde bir işlem gerçekleştirmeden önce o modelin ilgili işlemin gerçekleştirilmesi açısından uygun olup olmadığının sınanması amacıyla kullanılmasıdır [5].

Bir diğer kabiliyeti ise, yine Şekil 4'de resmedilmiş olan kaynak kod üretilmesi işlemidir. Bu kabiliyet oAW'nin xPand bileşeni sayesinde karşılanmaktadır. Yine aynı adla anılan gramer kullanılarak modelden kaynak koda çevrimini tarif eden deyimler ile modelden kaynak kod üretilmesi mümkün olabilmektedir. xPand dili, sağladığı AOP (Aspect-Oriented Programming) özelliği ile kod üretimi amacıyla ayrı ayrı tanımlanmış olan modelden koda çevirim deyimlerinin tanımlanmış kesim noktalarına (point-cut) göre belli bir sırada ve birbirini etkilemeden tanımlanmasını mümkün kılabilmektedir. Bu sayede örneğin belli bir sınıf model elemanından C++ kaynak kod üretecek şekilde başlık ve gerçekleştirim dosyalarını üreten ayrı bir çevirim betik dosyası, öznitelikleri (attributes) üreten ayri bir çevrim betik dosyası ve metodların giriş ve çıkışlarında log mesajı üretecek komutları üreten ayrı bir çevrim betik dosyası üretilebilir ve bu dosyalar birbirlerinden bağımsız ancak ve ancak bir tek ana betik dosyasına bağımlı olarak geliştirilebilir ve çeşitlendirilebilirler [5].

Sağladığı tüm bu bileşenler bir oAW akışında öncelikle model dosyalarının belli bir kapsüle (slot) okunmuş olması sağlandıktan sonra değişik bileşenler ve bu bileşenlerin sağladığı değişik servisler ile istenen sırada işletilebilirler. Bu özelliği ile oAW oldukça geniş bir kullanım çeşitliliğine imkan sağlar.

7. Elektronik Harp Benzetim Senaryosu Hazırlama Yazılımı Geliştirilmesi

Büyük bir proje kapsamında Elektronik Harp Benzetim Senaryosu Hazırlama Yazılımı'na ihtiyaç duyulmuştur. Bu yazılımın amacı, adından da anlaşılacağı üzere, özel bir elektornik harp benzetimi için benzetim senaryosu hazırlanmasına imkan sağlamaktır. Geliştirilecek olan yazılımın üretmesi gereken senaryo formatı daha önce yapılan arayüz tanımlamama çalışmaları sırasında tasarlanmıştır ve görev bilgisayarında çalışacak olan benzetim yazılımı da bu senaryo formatına göre geliştirilmektedir. Oldukça kısıtlı bir süreç dahilinde geliştirilmesi gereken senaryo hazırlama yazılımı için nasıl bir yöntem izleneceğine dair çalışmalar yapılmıştır. MGYM'nin bu iş için uygun olduğuna karar verilmiş; açık kaynaklı ve ücretsiz Eclipse ürünleri kullanılarak böyle bir çalışmanın yapılabileceği görülmüştür. Yapılan araştırmalar ve prototip üretme çalışması sonrası oAW, EMF ve RCP mimarileri kullanarak çalışmaya başlanmasına karar verilmiştir. Şekil 6'de izlenen yazılım geliştirme süreci sonucu üretilen çıktılar ve kullanılan yazılım çerçeveleri gösterilmektedir.



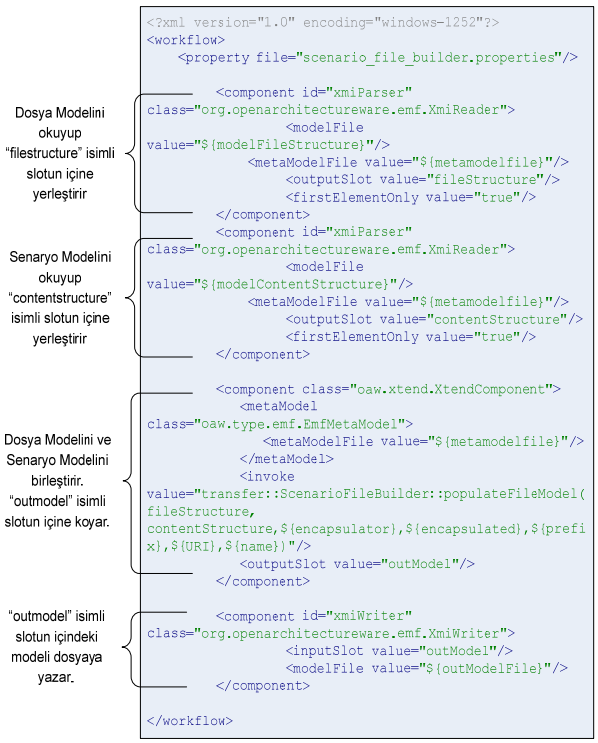
Şekil 6. Süreç ve Üretilen Çıktılar

Elektronik Harp Benzetim Senaryosu Modeli ile bu modelin içeriğini barındıracak genel amaçlı dosya modeli ayrı ayrı tasarlanmakta ve en son aşamada da birleştirilerek Senaryo Dosyası modeli elde edilmektedir. Bahse konu modellerin birleştirilmeleri amacıyla oAW'ın xTend bileşeninden yararlanılmaktadır.

Böyle bir gereksinimin ortaya çıkmasının sebebi yazılım geliştirme faaliyetlerinin gereksinim duyduğu veri yapıları ile etkinlik alanında ihtiyaç duyulan veri yapılarının birbirinden olabildiğince bağımsızlaştırılma tasarım kararı olmuştur. Bu ihtiyacı örneklemek gerekirse, senaryo içeriğinin kılıflandığı bir dosya yapısındaki değişiklikler ile senaryo yapısındaki değişiklikler (konfigürasyonlar) birbirlerinden bağımsız olarak izlenebilmekte ve bu değişiklikler birbirinden bağımsız olarak tasarlanabilmektedir. Bu model kullanılarak üretilen dosyayı okuyacak bir uygulama, dosya içeriğinin sürümünü senaryo modelinin böyle bir bilgi içermesine gerek kalmaksızın senaryo modelini kılıflamış olan dosya yapısını okuyarak elde edebilecektir. Bu dosya yapısının

geliştirilen yazılım genelinde tüm etkinlik alanı modellerini kılıflaması ile elde edilecek dosya yapılarında da bir standartlaşma sağlanmış olacaktır.

Bu modelleri birleştirme ihtiyacını karşılamak üzere aşağıdaki oAW akışı tanımlanmıştır.

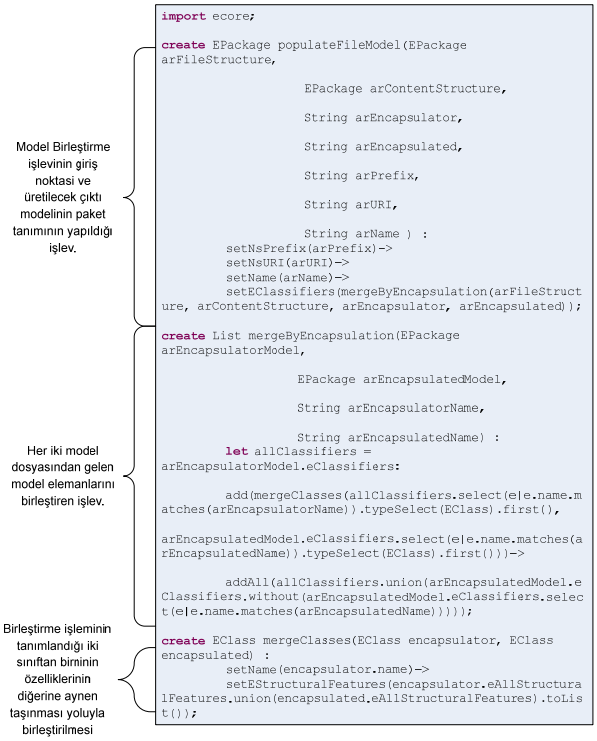


Şekil 7. oAW Akış Betiği

Şekil 7'de gösterildiği üzere oAW akışında sırasıyla önce birleştirilecek model dosyaları XmiReader bileşeni ile okunarak ilgili kapsüllere yerleştirilmekte, ondan sonra da birleştirme işlemini yapmak üzere XtendCompoenent bileşeni çalıştırılmaktadır. Son olarak da üretilen model dosyası XmiWrite bileşeni kullanılarak ilgili çıktı dosyası üretilmektedir.

Birleştirme işlemini yerine getiren populateFileModel işlevi de Şekil 8'de anlatılmaktadır.

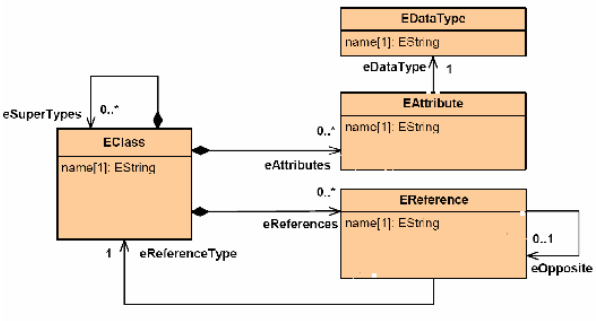
Şekil 8'de gösterilen işlevler ile sırasıyla model elemanlarını içeren paket oluşturulmakta, daha sonra iki modeldeki elemanlar bu paket altında birleştirilmekte ve son olarak da birleşime parametre olarak geçirilmiş kılıflanacak model elemanına ait özniteliklerin kılıflayacak elemana aktarılmasıyla birleştirme işlemi tamamlanmaktadır. Dolayısıyla populateFileModel işlevine, model birleştirme işlevinin karşılıklı olarak hangi iki model elemanı üzerinden gerçekleştirileceği ilgili model elemanlarının isimlerinin argüman olarak geçirilmesiyle sağlanmaktadır.



Şekil 8. xTend Çevrim Betiği

oAW servisleri ile tamamlanan modelleme aşaması ile takip eden bölümlerde detayları anlatılacak Elektronik Harp Benzetim Senaryosu Hazırlama Yazılımı aracının üretilmesi için ihtiyaç duyulan model üretilmiş olmaktadır.

EMF ile yazılım geliştirmenin ilk aşaması Şekil 4'de de görüldüğü gibi platform bağımsız bir model tasarlamaktır. EMF'de platform bağımsız model oluşturmak için *ecore* isimli dil kullanılır. Ecore, Object Management Group'un MOF 1.4 isimli standardından esinlenilerek geliştirilmiş ve daha sonra MOF 2.0 EMOF standardına göre uyarlanmıştır [8]. Şekil 9'de ecore meta-modelinin temel elemanları görülebilir.



Şekil 9. Ecore meta-modeli

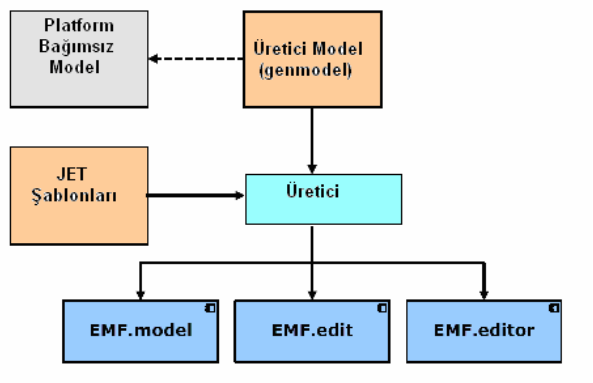
Ecore modeli, Ecore Editörü veya XML kullanarak tasarlanabilir. EMF modeli ise ecore modelden üretilebileceği gibi XSD, Java veya UML kullanılarak oluşturulmuş modeller de otomatik olarak EMF modellere dönüştürülebilir. Geliştirdiğimiz projede, benzetim senaryosunu kapsayan bir XSD model hazırlanması tercih edilmiştir. Geliştirilen XSD dökümanın başka amaçlar içinde kullanılması planlandığı için bu yöntem tercih edilmiştir.

XSD (XML Schema Definition) [9], XML dökümanlarının yapısını, yani elemanlarını ve elemanlarının özelliklerini tanımlayan bir dildir. Elektornik Harp Benzetim Senaryosu için geliştirilen XSD dökümanı, senaryonun elemanlarını (silahlar, ortam bilgileri gibi), elemanların özelliklerini (hız, konum gibi) ve elemanlar arası hiyerarşik yapıyı içeren bir dökümandır. XSD hazırlanması sırasında Eclipse Web Tools Platform'un içinde yer alan XML Schema Editor isimli bir editör kullanılmıştır [10].

Platform bağımsız modelinin tasarlanmasından sonraki aşama üretici modelin oluşturulmasıdır. Üretici model, platform bağımsız modele ek olarak, üretilecek kod paketlerine dair bilgileri ve üretilecek editörün bazı temel özelliklerine dair bilgileri içerir. Benzetim senaryosu hazırlaması yazılımının üretici modelini üretilirken yaptığımız en önemli seçim, üretilecek editör kodunun Eclipse RCP çerçevesine uygun olarak üretilmesini belirtmek olmuştur.

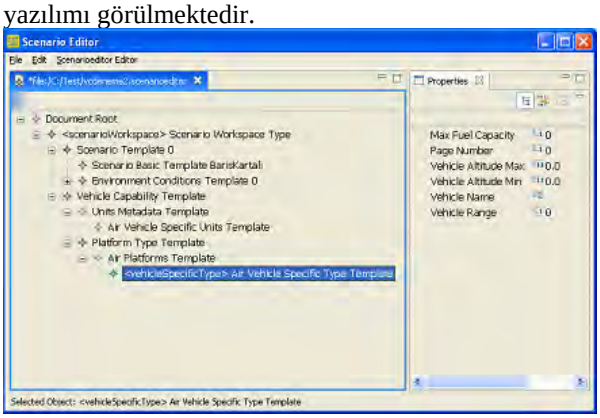
EMF, kod üretme süreci sonunda EMF.model, EMF.edit ve EMF.editor isimli üç adet eklenti üretilir. Bu eklentilerden EMF.model, platform bağımsız model bilgisini taşıyan bir eklentidir. EMF.edit ise görsel koddan bağımsız olarak adaptörleri ve içerik sağlayıcıları içeren bir eklentidir. EMF.editor ise görsel kodu içerir. Üretici modelde yapılan RCP uyumlu görsel arayüz üretilmesi seçiminin sonucu olarak, benzetim senaryosu hazırlama yazılımının görsel kodu RCP mimarisine uyumlu şekilde üretilmiştir.

Kod üretme işlevini Java Emitter Template (JET) isimli mekanizma yerine getirir. Şablon temelli olarak çalışan JET, Java kodu üretme ve birleştirme işlevi görür. JET, JSP diline benzer özellikler taşır ve zaten JET, Tomcat JSP derleyici yazılımı esas alınarak geliştirilmiştir. Geliştirdiğimiz yazılımda, EMF'nin standart kod üretme kabiliyetleri ve şablonları yeterli olduğu için özel bir geliştirme yapılmamıştır. EMF, kod üretme mekanizması Şekil 10'de gösterilmiştir.



Şekil 10. EMF Kod Üretme Sistemi

EMF tarafından üretilen EMF.model ve EMF.edit eklentileri ihtiyaçlarımıza büyük oranda cevap verdiği için bu eklentiler ile ilgili herhangi bir ek geliştirmeye gerek duyulmamıştır. Ancak görsel arayüzü içeren EMF.editor eklentisi ihtiyaçlarımıza tam olarak cevap vermemektedir. Dolayısıyla EMF.editor paketi üzerinde kapsamlı bir geliştirme çalışmasına ihtiyaç duyulmuştur. Birçok yeni özellik eklenerek daha esnek ve işlevsel bir görsel arayüz oluşturulmuştur. Şekil 11'de geliştirilen benzetim senaryosu hazırlama yazılımı görülmektedir.



Şekil 11. Benzetim Senaryosu Hazırlama Yazılımı

8. Sonuçlar

Bu projede geliştirilen benzetim senaryosu hazırlama yazılımı, temelde belli bir modele uygun

senaryolar geliştirilmesine olanak sağlamaktadır. İş dünyasında benzer şekilde, işlevi belirli bir veri yapısına veya belirli bir modele uygun veri kümeleri hazırlamak olan bir çok yazılım geliştirilmiştir ve geliştirilmektedir. Bu tür yazılımlar, benzer işlevleri yerine getiren ama gerek görsel arayüz gerekse ek işlevler bakımından birbirlerinden ayrılan yazılımlardır. Belirli bir modele uyumlu veri kümelerinin oluşturulması, saklanması ve güncellenmesi gibi ortak işlevleri yerine getiren bir yazılım parçasının otomatik olarak üretilebilmesi, bu tür yazılımların geliştirilme sürecini oldukça hızlandıracaktır. Otomatik üretilen kod parçaçıkları da uzmanlar tarafından geliştirilmiş ve defalarca test edilmiş olduğu için, üretilen yazılımın hata oranı oldukça düşük ve kalitesi oldukça yüksek olacaktır. Projemizde kullandığımız ve bu bildiride de tanıtmaya çalıştığımız EMF ve oAW çerçeveleri, yaygın olarak kullanılan, yazılım geliştiriciye birçok özellikler sunan, açık-kaynaklı ve ücretsiz olarak dağıtılan yazılım çerçeveleridir.

Senaryo modelinin daha önceden hazırlanmış olması ve zaman kısıtları nedeniyle başvurduğumuz MGYM yaklaşımı, bizim için oldukça tatminkar sonuçlar üretmiştir. Ücretsiz ve açık kaynaklı olarak dağıtılan Eclipse ürünleri de belirli bir olgunluğa erişmiş ve yazılım geliştiriciye oldukça kolaylık sağlayan ürünlerdir. Bu ürünler ve izlediğimiz yöntem oldukça kısa bir zaman sürecinde amaçladığımız yazılımı başarıyla geliştirmemizi sağlamıştır. Bu projede elde edilen bilgi birikiminin ve kazanılan süreçlerin bundan sonraki çalışmalarda oldukça yararlı olacağı düşünülmektedir.

9. Referanslar

[1] Stephen J. Mellor, Kendall Scott, Axel Uhl, Dirk Weise, *MDA Distilled. Principles of Model Driven Architecture*, Addison-Wesley Professional, 2004

[2] MDA Guide v1.0.1, Object Management Group, 2003

[3] http://www.eclipse.org

[4] Sherry Shavor., Jim D'Anjou., Scott Fairbrother., Dan Kehn , *The Java Developer's Guide to Eclipse*, Addison-Wesley Professional, 2003

[5] http://www.openarchitectureware.org

[6] http://en.wikipedia.org/wiki/Rich_Client_Platform

[7] http://www.topcased.org/

[8] http://www.omg.org/mof

[9] http://www.w3.org/XML/

[10] http://www.eclipse.org/webtools/

Model-Güdümlü Mimari Kullanılarak Bir Konum Sunucusu Yazılımının Geliştirilmesi

Fırat Yeşilürdü[1], Banu Diri[2]
[1]*Telenity İletişim Sistemleri A.Ş. firaty@telenity.com*
[2]*Yıldız Teknik Üniversitesi, Elektrik-Elektronik Fakültesi, Bilgisayar Müh. Bölümü*
banu@ce.yildiz.edu.tr

Özet

Model kavramını, yazılım geliştirmenin temel öğesi olarak kabul eden Model-Güdümlü Mimari (Model-Driven Architecture - MDA), uygulamaların geliştirilmesi için yeni bir yaklaşım sunmaktadır. Tamamlanmış bir Model Güdümlü Mimari (MGM) uygulaması; eksiksiz bir platform-bağımsız UML modeli, uygulama geliştiricinin desteklemeye karar verdiği platformlara ait bir veya daha fazla platform-bağımlı model ve tamamlanmış gerçekleştirmelerden oluşmaktadır.

MGM, gerçekleştirme detayları ile iş fonksiyonlarını birbirinden ayırmaktadır. Böylece her yeni teknoloji ortaya çıktığında, uygulama veya sistemin işlevini ve davranışlarını tekrar modellemeye gerek kalmamakta, yeni ve farklı teknolojileri desteklemek kolaylaşmaktadır.

Bu çalışma, MGM ve ilgili kavramları açıklamak ve bunları örnek bir yazılım projesi üzerinde hayata geçirmek amacını taşımaktadır. Örnek proje olan Konum Sunucusu yazılımı geliştirilirken, günümüzün MGM modelleme araçlarının elverdiği ölçüde, kodun mümkün olduğu kadar büyük bir kısmı yaratılan platform bağımsız model üzerinden otomatik olarak oluşturulmuş ve mümkün olduğu kadar az bir kısmı ise elle kodlanmıştır.

Abstract

Model-Driven Architecture (MDA) presents a new approach to application development by assuming 'model' the central element of software development. A complete MDA application consists of a complete platform-independent UML model, one or more platform-specific models and complete implementations that the application developer decided to support.

MDA separates the implementation details from the business functions. Thus, there is no need to model the functions and behavior of the application or system again whenever a new technology is introduced and it becomes easier to support new or different technologies.

This work has the aim to describe MDA and related concepts and use them on an example software project. While the example software project, Location Server is developed, as much as possible part of the code has been generated automatically from the platform-independent model as today's MDA tools allow and minimum coding has been done manually.

1. Giriş

Günümüzde yazılım geliştirmede model kullanım oranı gittikçe artmaktadır. Model, soyutlama seviyesinin yükselmesini sağlamakta ve yazılım geliştirmedeki maliyet, kalıcılık ve kalite faktörlerini pozitif yönde etkilemektedir [2].

Yazılım geliştirme yöntemlerinin tarihçesine bakıldığında, günümüze yaklaştıkça, yöntemlerin soyutlama seviyesinin sürekli yükseltmekte olduğu görülmektedir. Makine-merkezli yazılım geliştirmeden, uygulama-merkezli yazılım geliştirmeye ve işletme-merkezli yazılım geliştirmeye geçiş süreci içindeki, nesne-yönelimli diller, bileşenler, tasarım örnekleri, ara katman yazılımı, bildirimsel belirtim, kontrat ile tasarım gibi kavramlar soyutlama seviyesini yükseltmeye yardımcı olmuştur [2]. Bunlarla birlikte yazılım geliştirmede modelin önemi de gitgide artmıştır.

Model Güdümlü Mimari (MGM), yazılım endüstrisinin model tabanlı sistemler geliştirme ihtiyacını karşılamak amacıyla ortaya konan bir yaklaşımdır [6]. Model Güdümlü Mimari yöntemleri henüz yaygın olarak kullanılmamakla beraber, araç desteğinin yavaş yavaş artması ile birlikte artık

tamamen model tabanlı sistemlerin geliştirilmesi mümkün olabilmektedir.

MGM, bir belirtimin arkasındaki temel iş mantığı ile, o belirtimin gerçekleştirildiği, belirli bir platformun kendine has özellikleriyle ilgili bilgileri birbirinden ayırmaktadır. MGM, bugünün yüksek oranda ağ üzerine kurulu, sürekli değişen sistemleri için aşağıdaki özellikleri sağlayan bir mimaridir [8]:

- **Taşınabilirlik:** Tekrar kullanımın artması, günümüz ve gelecekte uygulama geliştirme ve yönetiminde masraf ve karmaşıklığının azaltılması.
- **Platformlar arası birlikte-işlerlik:** Birden fazla gerçekleştirme teknolojisi üzerine kurulmuş standartlar çerçevesindeki kesin metotlar ile, her gerçekleştirmenin aynı iş fonksiyonlarını yerine getirmesi.
- **Platform bağımsızlığı:** Henüz ortaya çıkmamış platformlar da dahil olmak üzere farklı platformlara geçişte yaşanacak zaman kaybı, maliyet ve karmaşıklığın azaltılması.
- **Alan özellikleri:** İş sahalarına ait özel modellerle farklı platformlarda, o iş sahasına ait yeni gerçekleştirmelerin hızlı bir biçimde hayata geçirilmesi.
- **Verimlilik:** Geliştirici, tasarımcı ve sistem yöneticilerinin rahat oldukları dil ve kavramları kullanabilmesi, böylece takımlar arası iletişim ve entegrasyonun kolaylaşması.

Bu çalışmada MGM'nin ve ilgili teknolojilerin yazılım endüstrisinin bugününe ve yarınına getireceği gelişmeler araştırılmıştır. Yazılım geliştirmede modeli temel öğe kabul eden bu yaklaşımın varolan gerçekleştirmeleri kullanılarak bir konum sunucusu uygulaması geliştirilmiştir. Bu uygulama geliştirilirken MGM'nin hangi aşamalarda ne gibi farklılıklar getirdiği de irdelenmiştir.

Konum Sunucusu, ağ operatörleri için katma değerli servislerin geliştirileceği uygulama geliştirme arayüzlerini, uygulama, abone ve gizlilik yönetimi hizmetlerini sunan bir platformdur. Bu çalışma kapsamında Konum Sunucusu kullanılarak geliştirilmiş bir örnek uygulama olan Çocuk Takip Servisi de gerçekleştirilmiştir.

Çalışmada odaklanılan nokta, MGM'nin büyük ölçekli ve dağıtık yapıdaki bir uygulama geliştirilirken sağladığı kolaylık ve zorluklardır. Bu çalışmada, mümkün olan en ileri aşamaya kadar platform bağımsız bir model oluşturulmaya çalışılmıştır.

2. Model-Güdümlü Mimarinin Uygulanması

Çoğu geliştirici, yazılım modellerine tasarım, program kodlarına ise geliştirme materyali olarak bakmaktadır. Birçok firma, model tasarımcısı ve programcı rollerini birbirinden ayırmaktadır.

Bunun sonucunda, modeller kuralsız olmakta ve bu yüzden makinenin işleyeceği bir yapıya sahip olamamaktadırlar. Programcılar bunları rehber ve tanım olarak kullanmakta, ancak üretime doğrudan katkı sağlayan şeyler olarak kullanamamaktadırlar.

MGM, makine tarafından işlenebilir formal modeller kullanmaktadır. Bu modeller üretim sürecinin doğrudan parçalarıdırlar. Böyle bir ortamda, model tasarımcısı ve programcının rolleri birbirine yakındır. Modelleme de bir programlama aktivitesidir.

MGM, bileşen tabanlı yazılım geliştirmeyi, tasarım örneklerini, ara katman yazılımlarını, bildirimsel belirtimi, soyutlamayı, çok katmanlı sistemleri ve Kontrat ile Tasarımı yeniden icat etmemekte, bunların üzerinde gelişerek, daha iyi çalışmalarına yardım etmektedir.

MGM'nin, soyutlama seviyesini 3. nesil programlama dillerinin üzerine çıkarması ile:

- Verimlilik artmaktadır, örneğin modeldeki bir komposizyon yapısı, 3. nesil programlama dilinin satırlarca koduna denk düşmektedir.
- Kalite artmaktadır, çünkü kod üreticiler iyi test edilmiş kodları tekrar oluşturmaktadırlar.
- Kalıcılık artmaktadır, çünkü anlamsal olan kısımlar, 3. nesil programlama dilleriyle alakalı kısımlardan ayrılmıştır [2].

MGM'de, bir uygulama için önce platform-bağımsız bir iş modeli hazırlanacak, sonra bu modelden platform-bağımsız bir tasarım modeli elde edilecektir. Daha sonra da bu modelden seçilen bir platforma (CORBA, Java, .NET, XMI/XML ve ağ tabanlı platformlar) ait modeller oluşturulacak, ve bu modeller hayata geçirilecektir [8].

2.1. Seçilen Araç ve Teknolojiler

Konum sunucusunun geliştirilmesi için kullanılabilecek ürünler olarak AndroMDA 3.1, Borland Together Architect 2006, IBM Rational Software Architect 6.0, iO-Software ArcStyler 5.1 ürünleri incelenmiştir.

İncelenen ürünler arasından, hem azami sayıda teknolojiyi desteklemesi açısından, hem de MGM prensiplerine en yakın ürün olması açısından, açık kaynak kodlu AndroMDA ürünü seçilmiştir.

Çalışmanın UML modellemesinde MagicDraw UML aracı kullanılmıştır [5].

Geliştirilen uygulama projesi bir *J2EE* uygulamasıdır. Mümkün olduğu kadar platform bağımsız olarak geliştirilen proje, bütün J2EE uygulama sunucularına (JBoss, BEA Weblogic vs.) kurulabilir durumdadır. Uygulamada, servis tabanlı *Spring Framework* ile konum servisleri *EJB session bean* olarak gerçekleştirilmiştir. Bu servisler harici uygulamaların dışarıdan erişebilmesi için web servisleri olarak sunulmaktadır. Web servislerinin gerçekleştirilmesi için *Apache Axis* kullanılmıştır.

Veritabanından da bağımsız olarak geliştirilen Konum Sunucusu gizlilik yönetimi ve uygulama detayları vb. bilgilere bir nesnel-ilişkisel veritabanı eşlemleme teknolojisi olan *Hibernate* vasıtası ile erişmektedir.

Konum Sunucusu'nun iç yapısının gerçekleştirilmesini sağlayan yukarıda adı geçen teknolojiler için AndroMDA'nın *Spring, WebServices, Hibernate* kartuşları kullanılmıştır.

Konum Sunucusu'ndaki web arayüzleri *Apache Struts* ve *Java Server Pages* ile gerçekleştirilmiştir. Burada iş proseslerini içeren kullanım durumu, aktivite şemaları hazırlanmış ve bunlar üzerinden web uygulamasını oluşturabilen AndroMDA *BPM4Struts* kartuşu kullanılmıştır [4].

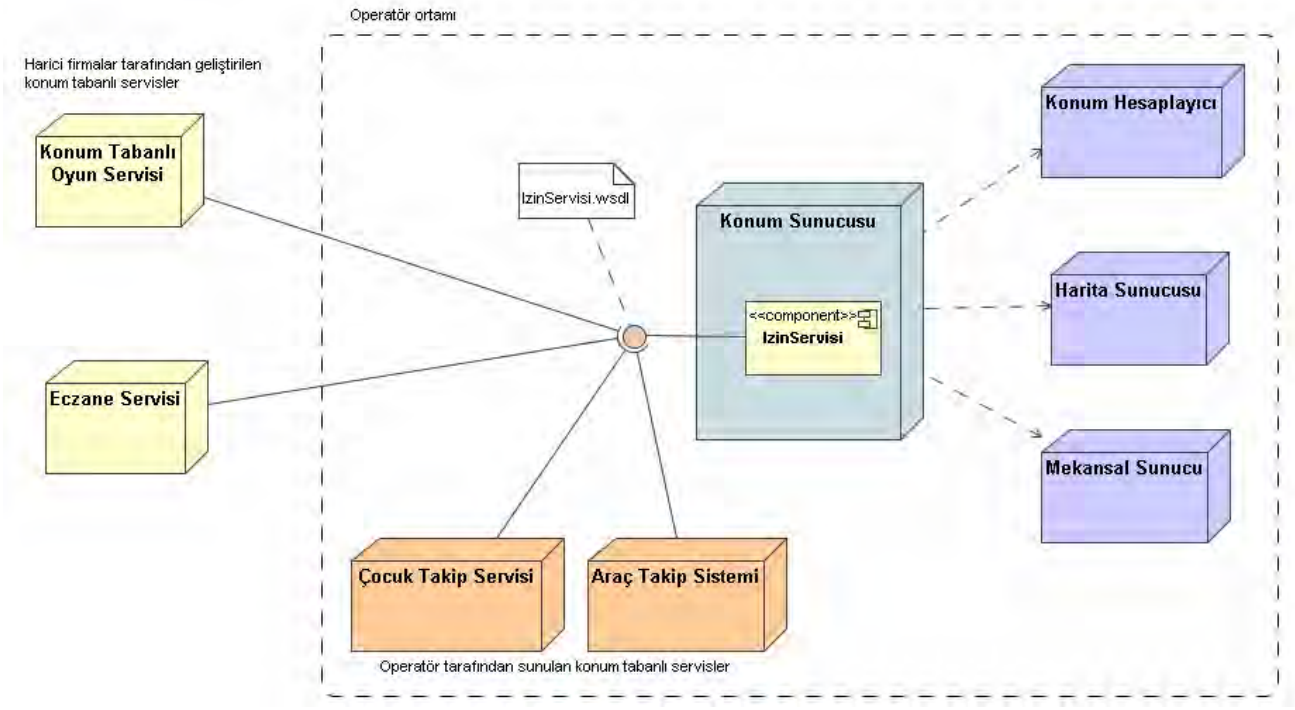
3. Konum Sunucusu

Ağ operatörleri, (örneğin GSM operatörleri) gün geçtikçe, konum bilgisini kullanan katma değerli servisleri hayata geçirmektedir. Bu servislerin ortak bir platform üzerinde geliştirilmesi, bunların hem geliştirilme sürecini kolaylaştıracak hem de operatörün bunları merkezi olarak denetleyebilmesini sağlayacaktır. Ayrıca konum gizliliği de merkezi olarak yönetilecektir. Çünkü konum gizliliği tam manasıyla sağlanmadığı sürece, aboneler ister istemez bu servisleri kullanmakta çekingen davranmaktadır [7].

Konum Sunucusu için geliştirilen uygulamalar operatör tarafından geliştirilip sunulacağı gibi, harici firmalar tarafından da geliştirilip, doğrudan müşterilere sunulabilir. Burada harici firmanın sunduğu uygulama doğal olarak uzaktan çalışmakta ve Konum Sunucusu ile Web Servisleri vasıtası ile haberleşebilmektedir. Aslında hem operatöre, hem de harici firmalara ait uygulamalar Konum Sunucusu'nun dışında bir yere kurulur ve bunların Konum Sunucusu ile tek iletişimleri Web Servisleri üzerindendir. Şekil 1'de Konum Sunucusu kurulum diyagramı görülmektedir.

Konum Sunucusu'na sorgulamalar, günümüzde birlikte işlerlik konusunda standartlaşmış olan Web servisleri vasıtasıyla gerçekleştirilebilmektedir. Sorgulamalar üç tiptir:

• **Konum sorgusu:** Sorulan abonenin koordinatını X,Y



Şekil 1. Konum Sunucusu Kurulum Diyagramı

olarak döner.

- **Adres sorgusu:** Sorulan abonenin adresini döner. Bu sorgu kendi içinde bir konum sorgusu gerçekleştirmekte, ardından elde ettiği koordinatı adrese dönüştürmektedir. Bu dönüşüm işlemine *reverse geocode* denmektedir.
- **Harita sorgusu:** Sorulan abonenin bulunduğu bölgenin haritasını bir URL olarak döner. Bu sorgu kendi içinde bir konum sorgusu gerçekleştirmekte, ardından elde ettiği koordinatı orta nokta olarak alan bir harita oluşturmaktadır.

Bir konum sunucusu, bu üç tip sorguyu da harici sistemleri kullanarak gerçekleştirir. Örneğin konum için operatöre ait bir GMLC (Gateway Mobile Location Centre) sistemi ile iletişim kurmakta, adres için bir mekansal (spatial) sunucuya sorgular gerçekleştirerek uygun bir adres oluşturmakta, harita içinse bir harita sunucusundan ilgili bölgenin haritasını istemektedir.

Aslında konum sunucusunun önemli bir özelliği, kullanıcıların bu çok çeşitli ve karmaşık yapıdaki sunucularla doğrudan iletişim kurmak yerine, Konum Sunucusu'nun sağladığı basit bir uygulama geliştirme arayüzünü kullanarak, zengin içerikli konum tabanlı sistemlerin kolayca gerçekleştirebilmesine imkan sağlamaktır. Ayrıca her bir harici sistemin farklı firmalar tarafından geliştirilmiş çeşitlerini destekleyerek, operatörlerin ve Konum Sunucusu'nu kullanacak programcıların işini oldukça kolaylaştırmaktadır.

Bu çalışma kapsamında geliştirilecek Konum Sunucusu bu üç tip sorguyu da simülasyon vasıtası ile gerçekleştirmektedir. Yapılan sorgular sonucunda önceden hesaplanmış koordinat, adres ve haritalar dönülmektedir.

Konum sunucusu, yönetici ve aboneler için iki farklı web arayüzüne sahiptir. Ayrıca Konum Sunucusu kullanımına örnek teşkil eden Çocuk Takip Servisi'nin de kendine ait bir arayüzü bulunmaktadır. Takip eden bölümlerde konum servislerinin ve abone web arayüzünün modellenmesi incelenecektir.

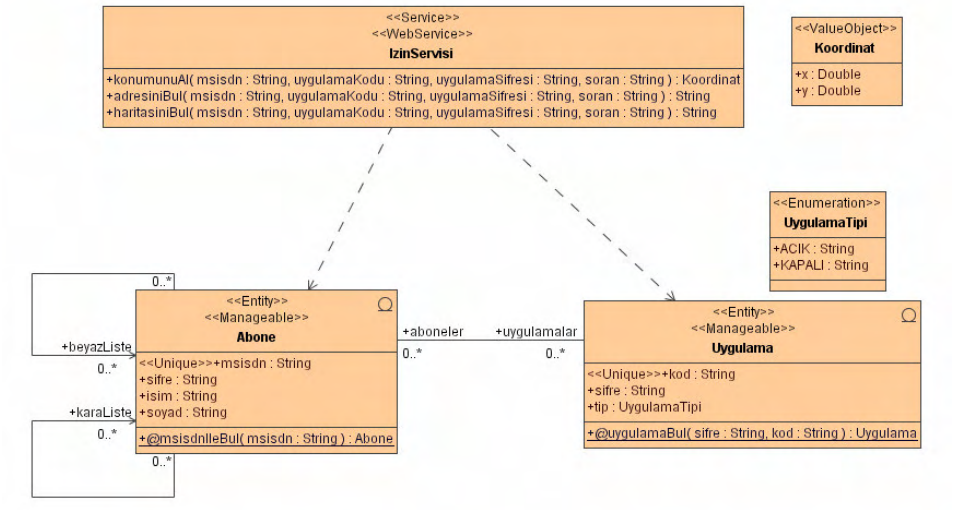
3.1. Konum Sunucusu Modelinin Oluşturulması

Projenin *mda* modülünde AndroMDA tarafından boş bir model yaratılmıştır. Bu model AndroMDA'ya ait UML profillerini kullanıma sunmaktadır. Öncelikle Konum Sunucusu'nun sunacağı servisleri ve temel varlıkları içeren sınıf diyagramı Şekil 2'deki gibi oluşturulmuştur [1].

Sınıf diyagramındaki *IzinServisi* adlı sınıf Konum Sunucusu tarafından sunulan üç servisi içermektedir. <<Service>> stereotipi, bu sınıfın *Spring* çatısına göre bir *EJB Session Bean* olarak oluşturulmasını, <<WebService>> stereotipi de bu servislerin web servis olarak dışarıya sunulacağını belirtir.

Abone sınıfı operatöre ait aboneleri temsil eden bir sınıftır. <<Entity>> stereotipi sınıfın veritabanında saklanacak (*Hibernate* ile) bir varlık olduğunu, <<Manageable>> stereotipi ise standart olarak kullanılan yarat-oku-güncelle-sil (CRUD: create-read-update-delete) servislerinin ve sunuş katmanında kullanılacak değer nesnesi (value object) sınıflarının otomatik olarak oluşturulmasını sağlar.

Abone sınıfının *msisdn* niteliği için verilmiş <<Unique>> stereotipi, bu niteliğin aboneleri birbirinden ayırt etmekte kullanılan nitelik olduğunu



Şekil 2. Konum Sunucusu Sınıf Diyagramı

belirtir. *beyazListe* ve *karaListe* birliktelikleri abonenin beyaz ve kara listelerindeki aboneleri göstermektedir. Bu birliktelikler ilgili sınıflarda sınıflar arasındaki ilişkilere ve veritabanı tabloları arasındaki kısıtlara dönüşecektir.

Abone sınıfındaki bir sorgu metodu olan *msisdnIleBul* metodu,

```
  context
Abone::msisdnIleBul(msisdn:String):Abone
  body msisdnIleBul: allInstances() ->
select (abone | abone.msisdn = msisdn)
```

OCL (Object Constraint Language - Nesne Kısıt Dili) ifadesini kısıt olarak barındıran bir metottur [3]. Bu metot, modelin işlenmesi sonrasında bir Hibernate sorgusuna dönüştürülecektir. Bu Hibernate sorgusu da çalışma zamanında bir SQL sorgusuna dönüştürülerek veritabanına gönderilecektir.

Uygulama sınıfı Konum Sunucusu'nda tanımlanacak uygulamaları temsil etmektedir. *UygulamaTipi*, <<Enumeration>> stereotipi ile uygulama sınıfının tip niteliğinin alabileceği değerleri sınırlamaktadır. Bu değerlerin veritabanına yazılmasından sayfalarda görüntülenmesine kadar her şey otomatik olarak yapılmaktadır. Yazılım geliştirici koda müdahale ederken, sadece ACIK ve KAPALI adlı sabit değerleri kullanmaktadır.

Abone ve *Uygulama* sınıfları arasındaki birliktelik, *Abone* sınıfından abonenin üye olduğu uygulamalara, *Uygulama* sınıfından uygulamaya üye olan abonelere ulaşmayı sağlamaktadır.

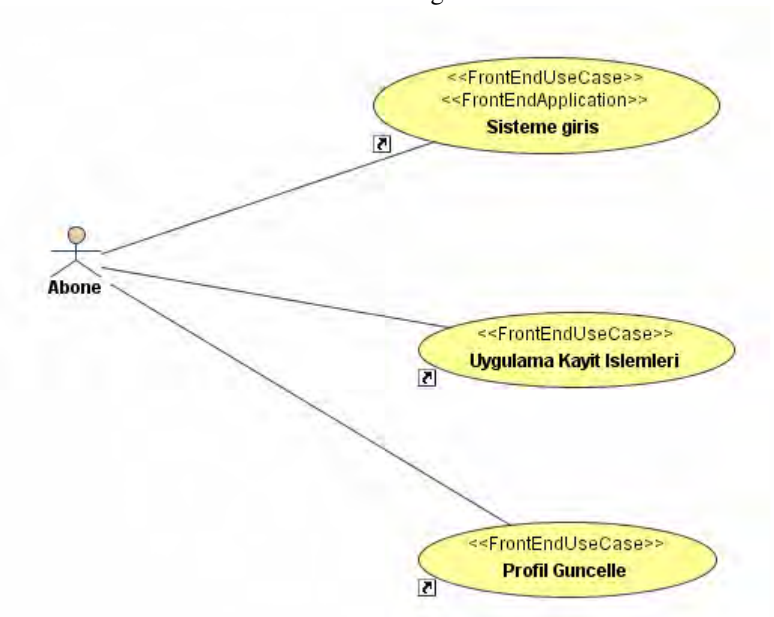
IzinServisi ile, *Abone* ve *Uygulama* sınıfları arasındaki bağımlılık ilişkisi, *IzinServisi*'nin bu varlıklara ait veri erişim nesnelerine (data access object) ulaşmasını sağlayacaktır.

3.2. Konum Sunucusu Abone Arayüzünün Tasarlanması

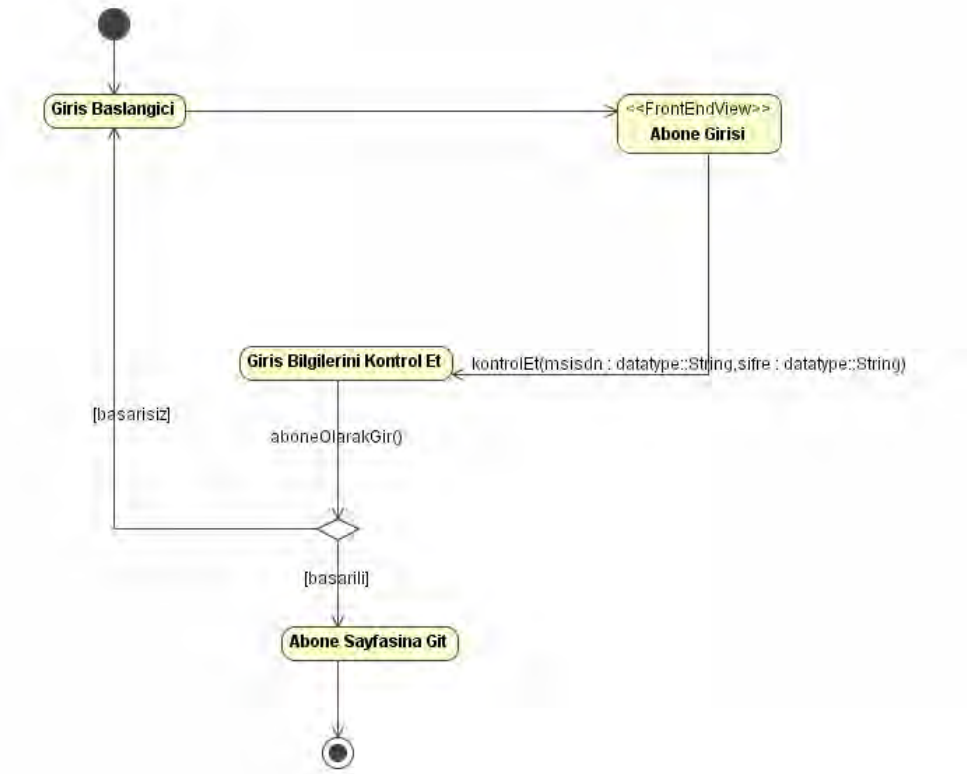
Abone arayüzünün tasarımına öncelikle Şekil 3'teki gibi bir kullanım durumu diyagramı hazırlanarak başlanmıştır.

Bu diyagramda *Abone* aktörünün sistemi kullanım amaçları gösterilmektedir. Her bir kullanım durumu <<FrontEndUseCase>> stereotipine sahiptir ve her biri ile bağlantılı aktivite diyagramları bulunmaktadır. <<FrontEndUseCase>> stereotipi *Sisteme Giriş* kullanım durumunun uygulamaya giriş noktası olduğunu belirtmektedir.

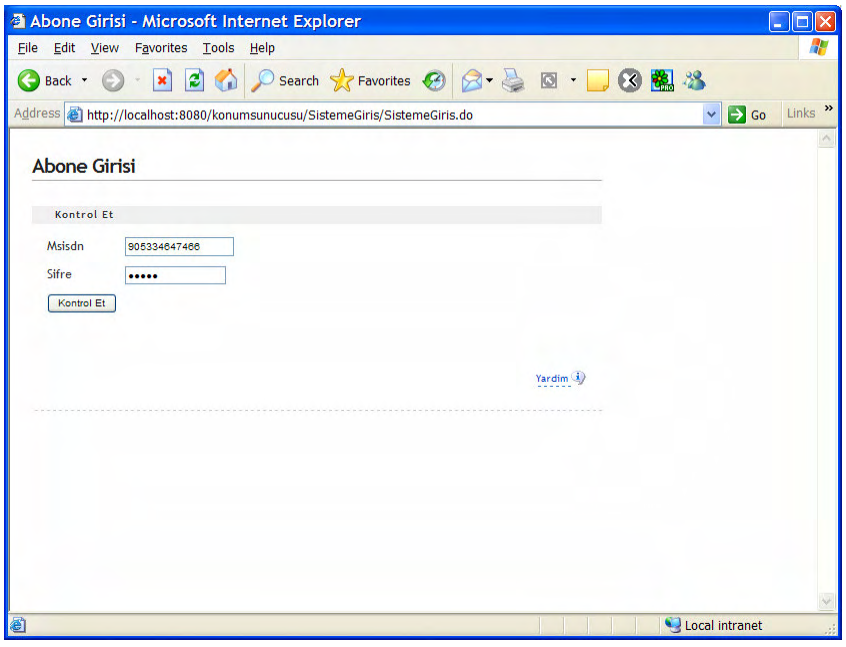
Şekil 4'te *Sisteme giriş* kullanım durumu için hazırlanan aktivite diyagramı görülmektedir. <<FrontEndView>> stereotipi kullanıcıdan giriş beklenen eylem durumları için kullanılır. Burada *Abone Girişi* eylemininden *Giriş Bilgilerini Kontrol Et* eylemine çizilen geçiş üzerindeki *kontrolEt* tetikleyicisi (trigger) kullanıcıdan *msisdn* ve *şifre* bilgilerinin sayfadaki metin kutuları ile alınmasını sağlamaktadır. <<FrontEndView>> stereotipli eylemlerden çıkan geçiş, sayfada bir gönder (submit) düğmesi olarak varolacaktır. Şekil 5'te diyagrama göre oluşan sayfa görülmektedir.



Şekil 3. Konum Sunucusu Abone Arayüzü Kullanım Durumları



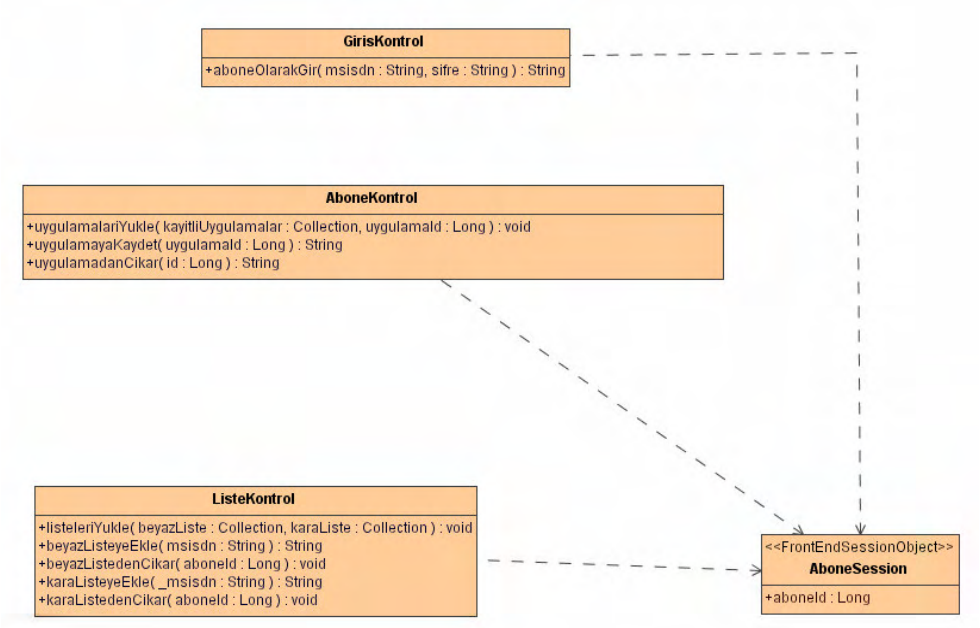
Şekil 4. Sisteme Giriş Aktivite Diyagramı



Şekil 5. Sisteme Giriş Sayfası

Her aktivite diyagramına ait bir kontrol sınıfı bulunmaktadır. Bu sınıftaki metotlar, sunuş ve iş katmanı arasındaki kontrol katmanı işlevini görmektedir. Şekil 6'daki sınıf diyagramında Konum Sunucusu Abone kontrol sınıfları görülmektedir. yazılmıştır. (andromda.presentation.action.warning. message=Msisdn-şifre geçersiz).

Şekil 7'de *Uygulama Kayıt İşlemleri* kullanım durumu için hazırlanan aktivite diyagramı, Şekil 8'de ise bu diyagrama göre oluşan sayfa görülmektedir.



Şekil 6. Konum Sunucusu Abone Kontrol Sınıfları

Aktivite diyagramındaki *aboneOlarakGir()* geçişi *GirisKontrol* sınıfındaki metodu çağırmaktadır. Kontrol sınıflarındaki metotların içi geliştirici tarafından doldurulacaktır. Sınıfların boş hali ve ilgili sayfa değişkenlerinin metot tarafından kullanılabilir hale getirilmesi AndroMDA tarafından otomatik olarak gerçekleştirilmektedir.

Üç kontrol sınıfı ile <<FrontEndSessionObject>> stereotipi eklenmiş olan *AboneSession* sınıfı arasında bağımlılık ilişkisi bulunmaktadır. Böylece kontrol sınıfları, oturum boyunca *aboneId*'yi muhafaza eden *AboneSession* sınıfına erişebilecektir.

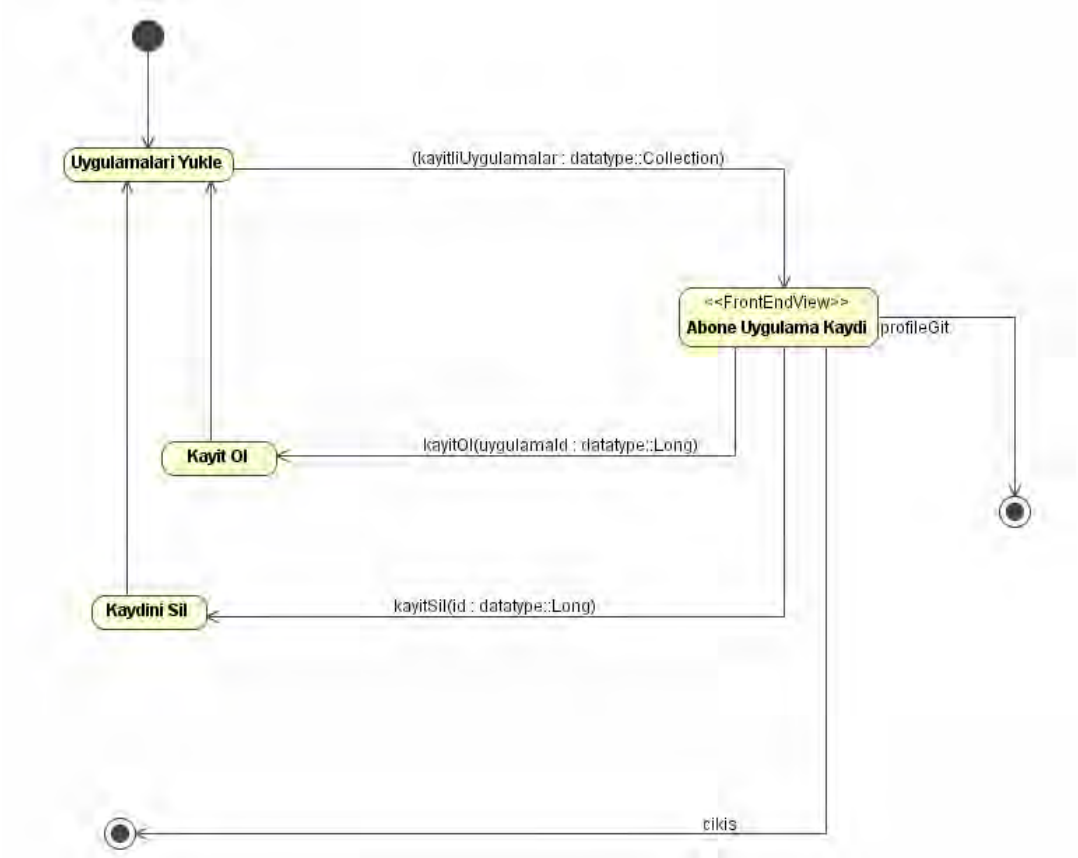
Eğer *msisdn* ve *şifre* geçerliyse abone sisteme girebilecek, değilse hata mesajıyla birlikte aynı giriş sayfasına dönecektir. Bu denetleme, aktivite diyagramında, kontrol metodundan geri dönülen değere göre bir karar noktası ve denetleyiciler (guard) ile sağlanmaktadır. *aboneOlarakGir()* metodunun geri dönüş değeri *basarili* ise *Uygulama Kayıt İşlemleri* kullanım durumuna yönlendirilmiş olan son duruma gidilecek ve *aboneId AboneSession*'a yazılacak, *basarisiz* ise *Giriş Başlangıcı*'na dönülecektir.

Msisdn-şifre'nin geçersiz olduğunu belirten hata mesajı, *[basarisiz]* geçişine bir etiketli değer olarak

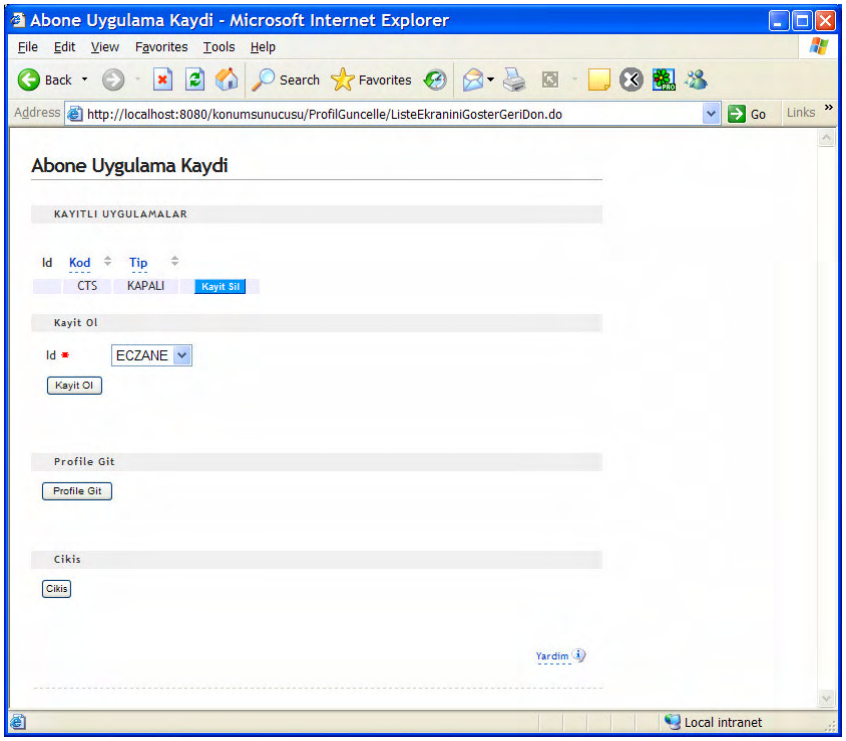
Uygulamaları Yükle eyleminde, aboneye ait uygulamaların yüklenmesi *AboneKontrol* sınıfındaki *uygulamalariYukle* metodunu çağıran bir ertelenebilir olay (deferrable event) ile sağlanmıştır. *Abone Uygulama Kaydi* eylemine giren geçişteki *kayıtlıUygulamalar* ekranda bir tablo olarak görünecektir. *Kaydını Sil* bu tablodaki seçime bağlı olarak çalışan bir eylemdir. Bütün bu detaylar, geçişler üzerinde yer alan tetikleyecilere ve parametrelerine etiketli değerler eklenerek sağlanmaktadır. *Kayıt Ol* ve *Kaydını Sil* eylemleri de kontrol sınıfındaki ilgili metotları çağırmaktadır.

Oluşturulan UML modeli, AndroMDA tarafından işlenerek, istenilen teknolojileri kullanan ve bir J2EE uygulama sunucusuna kurulabilir durumdaki çok katmanlı bir projeye dönüşmüştür. Geliştirilen uygulamada, farklı katmanlar için, modelden otomatik oluşturulan kodun, toplam koda yaklaşık olarak oranı aşağıdaki gibidir. Kalan kısımlar elle kodlama gerektirmiştir.

- Kalıcılık (EJB/Hibernate): 100%
- İş (EJB): 30%
- Web Servisleri (Axis): 100%
- Sunuş (Struts): 80%



Şekil 7. Uygulama Kayıt İşlemleri Aktivite Diyagramı



Şekil 8. Uygulama Kayıt İşlemleri sayfası

4. Sonuç

Yazılım projelerinde değişim kaçınılmazdır. Bir yazılım projesi, geliştirilirken de değiştirilebildiği gibi, kurulup kullanılmaya başlandıktan sonra gelen taleplere göre ve teknolojik ilerlemelere göre de değişime uğrar. Değişim ne kadar az emekle gerçekleştirilebiliyorsa yazılım o kadar esnek demektir. Bu açıdan bakıldığında, esneklik orta ve uzun vadede bir yazılımın kalitesini belirleyen önemli faktörlerden birisi haline gelmektedir.

Kötü tasarımlar sonucunda, yeni talepleri karşılayamaz hale gelen yazılım projelerinde kodun en baştan yazılması veya bir teknolojiden diğerine geçildiğinde projeye neredeyse en başından başlanılması, yazılım firmalarının sık sık karşılaştığı sorunlardandır.

Konum Sunucusu uygulama projesinde MGM'nin kullanımı ile, uygulamanın geliştirilme süresi ve yazılım kalitesi açısından oldukça tatmin edici sonuçlar elde edilmiştir. Yazılım katmanlarının kesin çizgilerle birbirinden ayrılarak tasarlanması ve bunun koda da aynı şekilde yansıtılması, klasik yöntemlerle kolay elde edilebilecek bir şey değildir. Bunun için yazılım geliştirme takımının oldukça disiplinli ve ahenkli biçimde çalışması gerekmektedir. Bu disiplin ve ahenk, proje başlangıcında sağlansa bile, ileri safhalarda karşılaşılabilecek zaman baskısı altında sağlanamayabilir.

MGM, uygulamanın prototipini oluşturmakta da oldukça iyi sonuç vermektedir. Prototipler, analiz ve tasarım aşamasında fark edilemeyen bazı gerçeklerin, projenin erken safhalarında fark edilmesinde önemli rol oynar. Çok çeşitli teknolojileri içeren Konum Sunucusu projesinde geliştirme aşama aşama gerçekleşmiş, bazı durumlarda varlık modeline dönülerek değişiklikler yapılmıştır. Bu değişiklikler model üzerinden kolayca yapılabilmiş, kod tekrar üretilerek değişiklikler projeye kolayca yansıtılmıştır.

Elle yazılan kodlar, kodun toplamına göre çok düşük orandadır. Bu kodlarda da otomatik üretilmiş sınıflar kullanıldığı için bütünlük bozulmamıştır.

Web tarafının aktivite diyagramları ile tasarlanması, alışılagelen yöntemlere göre farklı olduğu için istenilen sonuçların alınması zaman almıştır. Ancak ilk yapılan Konum Sunucusu'nun web arayüzünün tasarımı bittikten sonra Çocuk Takip Servisi'nin tasarımı çok daha kısa sürede gerçekleştirilmiştir.

MGM araçları ticari amaçla kullanılacaksa, yazılım firmasının kendi kurumsal mimarisini oluşturması en doğru yaklaşım olacaktır. Örneğin firmanın, ürünlerinde ortak bir sunuş katmanı tarzını uygulaması, MGM yaklaşımı ile verimli bir şekilde gerçekleştirilebilir.

Yazılım mimarları, firmalarında kurumsal bir mimarinin geliştirilmesini sağlamalıdır. MGM ile yazılım evleri, elde ettikleri tecrübeleri kurumsal bir mimari halinde toplayarak, birçok projede doğrudan kullanabilecek, yeni teknolojilerin kullanılması veya bir projenin bir teknolojiden diğerine geçirilmesi kolaylaşacaktır.

5. Referanslar

[1] Eriksson, H., Penker M., Lyons, B. ve Fado, D., "UML 2 Toolkit", Wiley Publishing, Inc., Indianapolis, Indiana, A.B.D., 2004

[2] Frankel, David S., "Model Driven Architecture: Applying MDA to Enterprise Computing", Wiley Publishing, Inc., Indianapolis, Indiana, A.B.D., 2003

[3] Warmer, J. ve Kleppe, A., "The Object Constraint Language: Getting Your Models Ready for MDA, 2nd Edition", Pearson Education, Inc., Boston, MA, A.B.D., 2003

[4] "AndroMDA 3.1 Reference Document", http://galaxy.androMDA.org/docs-3.1

[5] No Magic, Inc., "MagicDraw 11.0 User's Guide", http://www.magicdraw.com/files/manuals/11.0/MagicDraw%20UserManual.pdf

[6] OMG, "MDA Guide V1.0.1", http://www.omg.org/cgi-bin/doc?omg/03-06-01

[7] Location Based Services, Telenity, http://www.telenity.com/les.php

[8] Soley, R., "Model Driven Architecture", http://www.omg.org/docs/omg/00-11-05.pdf

Anlamsal Web Tabanlı Etmen Sistemlerinin Geliştirilmesinde Model Tabanlı Yaklaşım

Arda Göknil[1], Geylani Kardaş[2], N. Yasemin Topaloğlu[1], Oğuz Dikenelli[1]
[1]Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, Bornova, İzmir, Türkiye
[2]Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, Bornova, İzmir, Türkiye
{arda.goknil, geylani.kardas, yasemin.topaloglu, oguz.dikenelli}@ege.edu.tr

Özet

Model Tabanlı Mühendislik farklı soyutlama seviyelerindeki modelleri kullanarak yazılım geliştirmede karşılaşılan sorunları gidermeyi hedeflemektedir. Yakın zamanda model tabanlı yaklaşımlar etmen tabanlı yazılım geliştirme alanında da kullanılmaya başlanmış ve bu alanda yürütülen araştırmalar yoğunluk kazanmıştır. Ancak Anlamsal Web ortamında çalışacak etmen sistemlerinin modellenmesini göz önüne aldığımızda yürütülen çalışmaların yeterli olmadığı ve henüz hiçbirinin model tabanlı tekniklerde anlamsal web bileşenlerini tanımlamadıkları görülmektedir. Bu çalışmamızda, Model Tabanlı Mühendislik kullanılarak Anlamsal Web yetenekli Çok-etmenli Sistemler'in geliştirilmesine ait yaklaşımımız tanıtılmaktadır. Bu kapsamda, Model Tabanlı Mühendislik'in bir uygulaması olan Model Tabanlı Mimari bağlamında Anlamsal Web tabanlı etmen varlıklarının nasıl tanımlandığı, bu tür Çok-etmenli Sistemler'in geliştirilmesi için Model Tabanlı Mimari'nin bileşenleri ve gereksinimleri incelenmiş ve ayrıca Anlamsal Web yetenekli bir Çok-etmenli Sistem'in temel varlıklarının tanımlandığı çekirdek bir etmen üst-modeli (metamodel) sunulmuştur.

1. Giriş

Bilgi ve ağ teknolojilerinde görülen ilerlemeler aynı zamanda yazılım sistemlerinin de giderek karmaşıklaşmasına neden olmaktadır. Bu karmaşıklılarla başa çıkmak için yazılım mühendisliği alanında çalışmalarda bulunan araştırmacılar yeni yazılım geliştirme yaklaşımları, metotları ve teknikleri öne sürmektedirler. *Model Tabanlı Mühendislik* farklı soyutlama seviyelerindeki modelleri kullanarak yazılım geliştirmedeki karmaşıklığı azaltmayı hedeflemektedir. Model tabanlı mühendisliği yerine getirmek ve bundan faydalanmak için farklı seviyelerde model dönüşümü sağlayacak yazılım araçlarının bulunması gerekmektedir. *Model dönüşümü,* farklı soyut seviyelerindeki modellerin otomatik eşlenmesi için bir veya daha fazla kaynak modelini girdi olarak alır ve gerekli bir dizi dönüşüm kuralına bağlı olarak yine bir veya daha fazla hedef modelini çıktı olarak ortaya koyar.

Yakın zamanda model tabanlı yaklaşımlar etmen tabanlı yazılım geliştirme alanında da kullanılmaya başlanmış ve belli başlı araştırma konularından biri haline gelmiştir [5] [8] [4]. Çok-etmenli Sistemler'in geliştirilmesinde mümkün olan en yüksek soyutlama seviyesinde çalışmak oldukça kritik bir öneme sahiptir. Çünkü iç karmaşıklıklarından, dağıtık yapılarından ve açık sistemler olmalarından dolayı Çok-etmenli Sistemler'in kod seviyesi detaylarını ortaya çıkarmak ve gözlemlemek neredeyse imkansızdır.

Anlamsal Web evrimi şüphesiz etmen araştırmalarına yeni bir vizyon getirmiştir. Bu *İkinci Jenerasyon Web*, Dünya Geneli Ağ'ı (WWW) web sayfası içeriklerinin ontolojiler kullanılarak yorumlanabileceği bir seviyeye taşımayı hedeflemektedir. Söz konusu yorumlamanın ve anlam çıkarsamaların otonom hesaplama birimleri – yani etmenler – tarafından insan kullanıcıları adına yerine getirileceği açıktır.

Anlamsal Web ortamının kendine özgü mimari varlıklarının ve farklı bir semantiğinin olduğu, bu ortam üzerinde çalışacak Çok-etmenli Sistemler modellenirken göz önünde bulundurulmalıdır. Bu nedenle etmen sistemleri için ortaya konan modelleme tekniklerinin ve geliştirme sürecinin bu yeni ortamı yeni üst-varlıklar (meta-entity) ve üst-yapılar (meta-structure) tanımlayarak desteklemesi gerekmektedir. Literatürde her ne kadar etmen sistemlerinin geliştirilmesi için model tabanlı yaklaşımlar tanımlayan çeşitli çalışmalar [4] [8] [9] [16] yer alsa da bu çalışmalarda etmenlerin Anlamsal Web ortamında

çalışmaları ve bu ortama ait sistem bileşenlerinin tanımlanması göz önünde bulundurulmamıştır.

Bu çalışmamızda, Model Tabanlı Mühendislik kullanılarak Anlamsal Web yetenekli Çok-etmenli Sistemler'in geliştirilmesi için yaklaşımımızı sunmaktayız. Bu kapsamda, Model Tabanlı Mühendislik'in bir uygulaması olan Model Tabanlı Mimari bağlamında Anlamsal Web tabanlı etmen varlıklarının nasıl tanımlandığı, bu tür Çok-etmenli Sistemler'in geliştirilmesi için Model Tabanlı Mimari'nin bileşenleri ve gereksinimleri incelenmiş ve ayrıca Anlamsal Web yetenekli bir Çok-etmenli Sistem'in temel varlıklarının tanımlandığı çekirdek bir etmen üst-modeli (metamodel) tanıtılmıştır.

Bildirinin ikinci bölümünde Model Tabanlı Mühendislik, üst-modelleme, model dönüşümleri, Anlamsal Web tabanlı Çok-etmenli Sistem mimarisi ve bunların bütünleştirilmesine ait genel bir bakış sunulmaktadır. Üçüncü bölümde önerdiğimiz etmen üst-modeli anlatılmaktadır –ki bu üst-model, Anlamsal Web yetenekli Çok-etmenli Sistemler'in geliştirilmesinde model tabanlı bir mimariyi dahil etmenin ilk adımıdır. Önerilen bu üst-modelden çalıştırılabilir modellerin türetilmesi konusu ise dördüncü bölümde yer almaktadır. Son bölümde elde edilen sonuçlar ve gelecek için hedeflenen çalışmalar yer almaktadır.

2. Anlamsal Web Tabanlı Etmen Sistemleri için Model Tabanlı Mühendislik

Anlamsal Web'in şu anki Web'in bir uzantısı olan, bilginin düzgün tanımlanmış bir anlama sahip olduğu ve insanlar ile bilgisayarların beraber çalışabildikleri bir web olarak çalışması hedeflenmektedir [25]. Günümüze kadar Web, veri ve bilgilerin otomatik olarak işlenebildiği bir ortamdan çok insanlar için doküman sağlayan bir ortam olacak şekilde geliştirilmiştir. Anlamsal Web ise ilgili otomatik işlemeyi gerçekleştirme amacına sahiptir. Doğal olarak bu otomatik işleme ortamında insan kullanıcıları adına faaliyetlerde bulunması gereken otonom yapıların bulunması gerekmektedir.

Günümüzde hem etmen araştırmaları hem de anlamsal web üzerine yapılan çalışmalar birbirini destekleyecek şekilde devam etmektedir. Çünkü yazılım etmenlerinin anlamsal web ortamlarında çalışacağına ve özellikle yetenekleri anlamsal olarak modellenmiş servislerle etmenlerin etkileşime gireceğine inanılmaktadır. Bu nedenle etmen sistemlerini geliştirmeye yönelik kullanılacak modelleme dillerinin ve onların ürünü olan etmen üst-modellerinin anlamsal web yapılarını da kapsayacak şekilde tanımlanması gerekmektedir.

Model Tabanlı Mühendislik (Model Driven Engineering – MDE) [12], yazılım sistemlerinin model tabanlı geliştirilebilmesi için ortaya konulan mühendislik ilkelerini kapsamaktadır. Anlamsal web tabanlı yazılım etmenlerinin geliştirilmesi için *Model Tabanlı Mühendislik* kapsamında ortaya konan bu ilkelerden yararlanılabilir. *Object Management Group'un (OMG)*, Kasım 2000 tarihinde *Model Tabanlı Mühendislik* yaklaşımının bir uygulaması olarak ortaya koyduğu *Model Tabanlı Mimari (Model Driven Architecture – MDA)* [20], etmen sistemlerinin model tabanlı geliştirilebilmesi için uygun bir alt yapı sunmaktadır. Literatürde etmen tabanlı yazılım sistemlerinin Model Tabanlı Mimari'ye uygun olarak geliştirilmesi için bazı öneriler [4] [5] [8] [24] sunulmuş olmasına rağmen bu önerilerden hiç biri anlamsal web tabanlı etmen sistemlerini desteklememektedir.

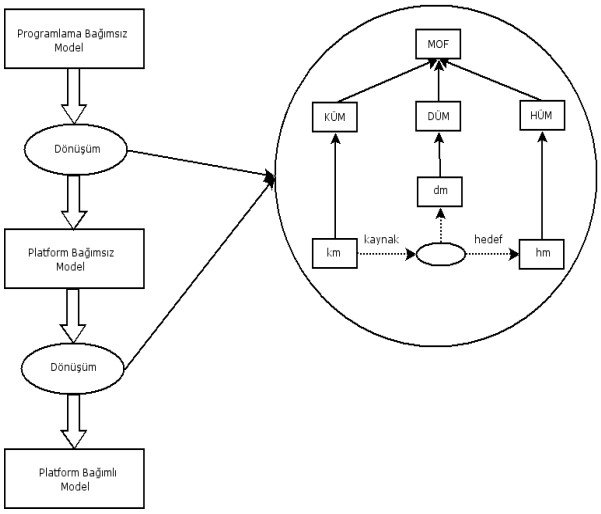
Model Tabanlı Mimari ile nesne modellerinin çalıştırılabilir bileşenlere ve uygulamalara dönüştürülerek yazılım sistemlerinin geliştirilmesi amaçlanmaktadır. Model Tabanlı Mimari kapsamında üç soyutlama seviyesi tanımlanmaktadır: Programlama Bağımsız Modeller (*Computation Independent Models – CIMs*), Platform Bağımsız Modeller (*Platform Independent Models – PIMs*) ve Platforma Özgü Modeller (*Platform Specific Models – PSMs*). Tüm bu soyutlama seviyelerinin en altında da kaynak kod yer almaktadır. Geliştirici, programlama bağımsız modelleri kullanarak sadece problem içerisinde yer alan varlıkları ve bu varlıklar arasındaki ilişkileri modelleyecektir. Bu soyutlama seviyesindeki bir modelde uygulama ile ilgili herhangi bir bilgi yer almamaktadır. Örneğin bir etmen sistemi için programlama bağımsız model, turizm alanı gibi anlamsal web etmenlerinin otel işlemlerine destek vereceği bir alanı tanımlayabilir. Ancak programlama bağımsız modellerden platform bağımsız modellere otomatik olarak geçebilmek için geliştirici, programlama bağımsız model üzerinde işaretleme işlemini yapmalıdır. Model dönüşüm tanımları ise programlama bağımsız modelde yer alan işaretlemelerden yola çıkarak bir alt seviyedeki platform bağımsız modeli oluşturacaktır. Örneğin, programlama bağımsız modelde yer alan varlıklar işaretlenerek platform bağımsız modelde etmenler ve anlamsal web etmenleri oluşturulabilmelidir. Bauer ve Odell [5] bir etmen sistemi için muhtemel programlama bağımsız modelleri belirtmişlerdir.

Platform bağımsız modeller, herhangi bir platformu temel almamasına rağmen programlanabilir yapıları içerisinde barındırmaktadır. Platform bağımsız

modellerden herhangi bir platforma özgü modele geçiş tanımlanabileceği gibi bu modellerden doğrudan platforma özgü kaynak kod da üretilebilir olmalıdır.

Anlamsal web tabanlı bir etmen sistemi için tanımlanacak olan platform bağımsız model, sisteme değişik bakış açılarından yaklaşan bir yapıda olmalıdır. Benguria [10], servis tabanlı mimariler için tanımlanacak platform bağımsız modellerde dört ana bakış açısı tanımlamıştır: *Bilgi (information aspect), Servis (service aspect), İşlem (process aspect)* ve *Servis Kalitesi (quality of service aspect).*

Anlamsal web tabanlı bir etmen sistemi için platform bağımsız model, belirli bir etmen çerçevesine ait olmayan yapıları ve bu yapılar arasındaki ilişkileri içermelidir. Bir etmen platformuna bağımlı olmayan bu yapılar, SEAGENT [21] gibi anlamsal web desteği sağlayan bir etmen sistemine ait platform bağımlı model içerisindeki yapılara dönüştürülebilir. Bu yaklaşımın en belirgin özelliği, platform bağımlı modellere dönüşümde otomatik olarak istenilen platforma ait model üretilebilmektedir. Şekil-1'de Model Tabanlı Mimari kapsamında geliştirme süreci ve modeller arasında tanımlanan dönüşüm adımları verilmektedir.



Şekil 1: Model Tabanlı Mimari için Model Seviyeleri ve Dönüşümler

Şekil 1'deki gibi üst modele dayanan dönüşümlerde kaynak model (km), dönüşümün sonunda elde edilecek hedef model (hm) ve dönüşüm modeli (dm) – dönüşüm kuralları tanımlanmış olmalıdır. Model dönüşüm işleminde öncelikle başlangıç modeli yani kaynak model içerisinde kaynak sorgular işletilir ve değiştirilmek istenen yapılar belirlenir. Daha sonra model dönüşümü için tanımlanmış olan dönüşüm kuralları, belirlenen bu yapılar üzerinde uygulanır. Dönüşüm kurallarının uygulanması sonucu üretilen hedef modelin hedef üst modeline (HÜM) uygun olması gereklidir. Böylece üretilen modelin doğruluğu, hedef üst model ile yapılan eşlemelerde sınanır. Şekil 1'de görülen kaynak ve hedef üst modeller, dönüşüme girecek kaynak modeller ile dönüşüm sonucu elde edilecek hedef modellerin bilgilerinin tutulduğu modellerdir. Bu dönüşüm tanımları ise değişik model dönüşüm dilleri [1] [2] [7] [18] kullanılarak ifade edilebilir. Şekil 1'de verilen Model Tabanlı Mühendislik için tanımlanan değişik modeller arasındaki dönüşümler ve dönüşüm mekanizması, etmen tabanlı modeller ve bu modeller arasındaki dönüşümler için kullanılabilir. Perini [4] tarafından ortaya konan çalışma etmen tabanlı sistemlerin Model Tabanlı Mimari kapsamında geliştirilebilmesi için tanımlanan dönüşümlere güzel bir örnektir. Perini [4], çalışmasında TEFKAT model dönüşüm dilini [18] kullanarak Tropos üst modelinde tanımlı plan yapılarını UML Activity modellerine dönüştürmüştür.

Şekil 1'den de görüleceği üzere modeller arası işbirliğini ve dönüşümü sağlamada önemli bir bileşen, üst modelleme (*meta-modeling*) kavramı ve üst modellerdir. Bir üst model (*meta-model*), belirli bir modelleme dilinde geçerli olan modelleri tanımlayan bir modeldir. Üst model gibi üst düzey tanımlamalar, modeller üzerinde çalışılmasını kolaylaştırırlar. Etmen tabanlı sistemlerin modellenmesinde ve bu etmen modellerinin birbirine dönüştürülmesi çalışmalarında etmen üst modellerinin tanımlanması önemli bir yer tutmaktadır. Bu kapsamda Gaia, Adelfe, PASSI [6] ve SODA [3] gibi etmen metodolojileri geliştirmek için çeşitli etmen üst modelleri önerilmiştir. Ancak bu etmen üst modellerinin bir çoğu sadece genel kavramları modellemek için kullanılmaktadır ve yine bu üst modeller, etmen geliştirme ortamları için tanımlanmış modeller değildir [16].

Önde gelen etmen üst modeli çalışmalarından biri de *FIPA* ile *OMG'nin (Object Management Group)* Etmen Özel İlgi Grubu'nun (Agent SIG) ortaklaşa yürüttükleri *Agent Class Superstructre Metamodel* [15] çalışmasıdır. Bu çalışma, bu grupların kendilerinin de belirttiği gibi başlangıç safhasındadır. Önerilen üst model; etmenleri, etmen rollerini, etmen gruplarını ve bunların birbirleri ile olan ilişkilerini tanımlamaktadır. Bu üst model, UML tabanlı olup aynı zamanda onun bir uzantısıdır. Etmen sistemleri için üst model çalışmaları yeni olmayıp ALAADIN üst modeli [13], bu üst model çalışmalarının öncülerindendir.

Diğer yandan MetaDIMA [26], Model Tabanlı Mimari yaklaşımından etkilenerek etmen sistem mimarileri ile etmen tabanlı metodolojiler arasındaki kopukluğu gidermeye çalışan model tabanlı bir diğer

projedir. Pavon [16] ise çalışmasında INGENIAS adını verdikleri etmen geliştirme metodolojilerini yeniden tanımlayarak model tabanlı geliştirme paradigmasına uygun hale getirmiştir.

Etmen ortamlarına anlamsal web desteği getirmeleri açısından etmen üst modeli çalışmalarını incelediğimizde henüz bu çalışmalardan hiçbirinin bu desteği verecek yapıları oluşturmadıkları ortaya çıkmaktadır. Bunun gelecek nesil çok-etmenli yazılım sistemlerinin Anlamsal web ortamlarında çalışması gerekliliği ve etmenlerin planları dahilinde yetenekleri anlamsal olarak tanımlanmış web servisleri ile etkileşimde olma gereksinimleri de düşünülürse önemli bir dezavantaj olduğu açıktır.

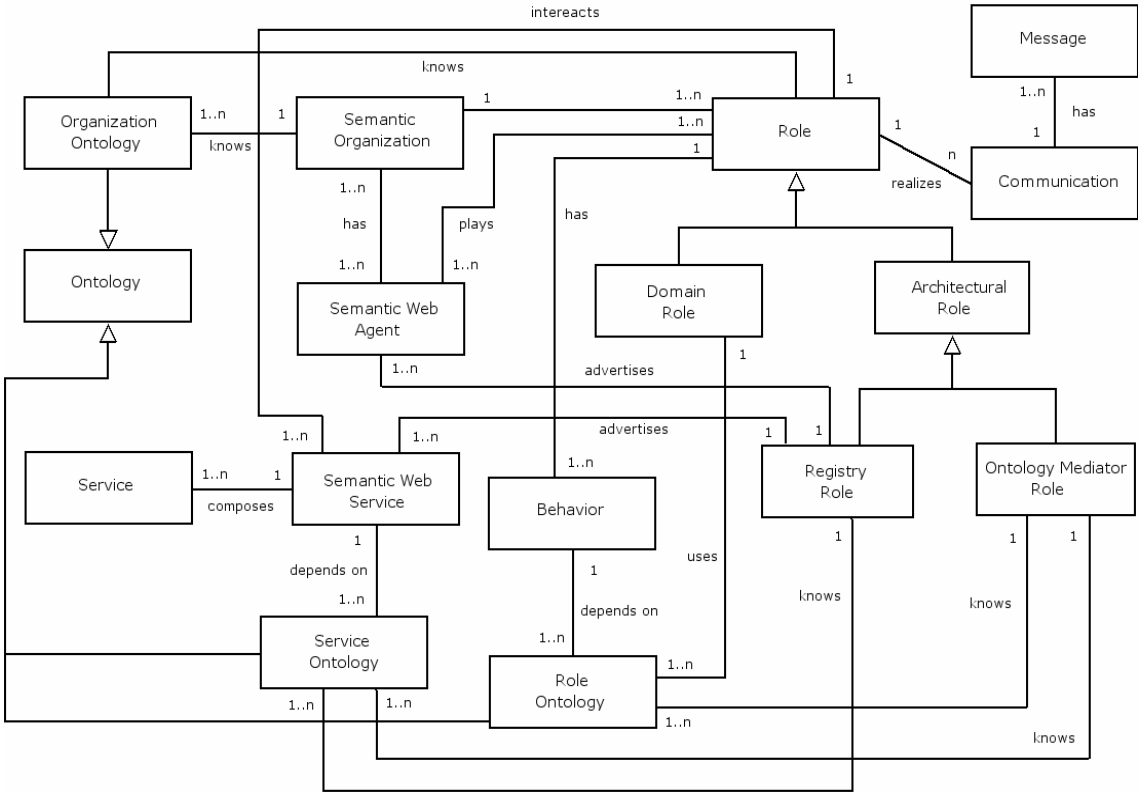
3. Anlamsal Web Üzerinde Çalışacak Çok-etmenli Sistemler İçin Bir Üst-Model

Anlamsal Web ortamında çalışacak etmen sistemleri için genel kavramları ve varlıkları (entity) tanımlayan bir üst model (metamodel) [11]'de tanıtılmıştır. Bu üst model, Model Tabanlı Mimari'ye dayalı olarak bu tip sistemlerin geliştirilmesi sırasında gereksinim duyulan Platform Bağımsız Model ve Platforma Özgü Model'lere ait varlıkların tanımlanmasında anahtar rol üstlenmektedir. Tanımlanan çekirdek üst model Şekil 2'de yer almaktadır. Açıklamalar sırasında anlam karmaşıklıklarını önlemek amacıyla model varlıkları, orijinal İngilizce adları ile kullanılacaktır.

Semantic Organization, *Semantic Web Agent*'larının oluşturduğu bir yapı olup etmenlerin organizasyonel rollerine dayalı olarak oluşturulur.

Bir *Semantic Web Agent*, ortamdaki hem diğer etmenler ile hem de anlamsal web servisleri ile etkileşime geçme yeteneğine sahip otonom bir varlık olarak tanımlanmaktadır.

Burada dikkat edilmesi gereken organizasyonun sadece etmenlerden oluştuğu ve etmen harici anlamsal yapılar içermediğidir. Sadece organizasyonel rollere yer verilmektedir ve bu roller üye etmenler tarafından oynanmaktadır. Organizasyonel rolleri göz önüne aldığımızda bir Anlamsal Web Etmeni aynı anda birden fazla organizasyonun üyesi olabilir. Bu demektir ki ***bir etmen aynı anda birden fazla rolü oynayabilir ve bir rol Semantic Organization içeriğinde (context) birden fazla Semantic Web Agent tarafından oynanabilir***.



Şekil 2: Anlamsal Web yetenekli Çok-etmenli Sistemler için çekirdek bir üst-model

Role varlığının mimari ve alan (domain) tabanlı rollerin görev tanımlarına bağlı olarak model içerisinde özelleşmesinin gerektiğine inanmaktayız. Bu nedenle model içerisinde bu üst-varlığın (meta-entity) iki alt-varlığı (sub-entity) yer almaktadır: *Architectural Role* ve *Domain Role*. Bir *Architectural Role*, organizasyonun içeriğinden bağımsız olarak platform içerisinde en az bir etmen tarafından üstlenilmesi gereken anlamsal web yetenekli çok-etmenli sistemler için zorunlu bir rolü tanımlamaktadır. Buna karşılık *Domain Role* etmen organizasyonu içeriğine özgü olup tamamıyla özelleşmiş bir iş alanı için oluşturulmuş bir *Semantic Organization*'ın gereksinimlerine ve görev tanımlamalarına dayalıdır.

Organizasyonda yer alan bir kısım etmenin [11]'de tanıtılan kavramsal mimarinin Mimari Servis Katmanı'nda tanımlanmış olan servisleri sağlamak amacıyla tanımlanmış rolleri oynaması gerekir. Bu nedenle üst-modelde *Architectural Role*'un iki özelleşmiş alt-varlığı da yer almaktadır: *Registry Role* ve *Ontology Mediator Role*.

Bir *Role* bir ya da daha fazla *Behavior*'u içermektedir. Bir *Semantic Web Agen*t'in görev tanımları ve ilgili görev işletim süreçleri *Behavior* varlıkları içerisinde modellenmektedir.

Oynanan rollere bağlı olarak etmenlerin çoğu zaman başka bir etmen ile iletişime geçmesi gerekmektedir. Üst-modeldeki *Communication* varlığı platformdaki iki etmen arasındaki bir etkileşimi tanımlamaktadır. Etkileşim önceden tanımlı etmen etkileşim protokolüne dayalı olarak gerçekleşir. Bir *Communication* varlığı bir ya da daha fazla *Message* varlığı içermektedir. Her bir mesajın içeriği *RDF (Resource Description Framework)* tabanlı bir anlamsal içerik dili ile ifade edilebilmektedir.

Bir *Semantic Web Service* üst-varlığı etmen servisi hariç olmakla birlikte yetenekleri ve etkileşimleri mevcut sistem içerisinde anlamsal olarak tanımlanmış herhangi bir servisi temsil etmektedir. Bir *Semantic Web Service* bir ya da daha fazla *Service* varlığını içermektedir. Her servis bir web servisi ya da gerçek hayat uygulamasında önceden tanımlı başka özel bir etkileşim protokolüne sahip bir servis olabilir. Ancak mutlaka ilgili servisin platformun etmenleri tarafından kullanılabilecek bir anlamsal arayüzünün olması gerekmektedir. Burada dikkat edilmesi gereken modelde *Semantic Web Agent'ları ile Semanic Web Service'leri arasındaki ilişkinin Role varlığı üzerinden sağlanmış olmas*ıdır. Çünkü ***etmenler organizasyon içerisinde tanımlanmış olan rollerine dayalı olarak anlamsal web servisleri ile etkileşimde bulunurlar***.

Diğer Anlamsal Web ortamları gibi anlamsal web yetenekli çok-etmenli sistemler de ontolojiler olmaksızın düşünülemezler. Bu nedenle önerdiğimiz üst-model bir *Ontology* varlığı ve onun ihtiyaç duyulan alt-varlıklarını içermektedir. Bir *Ontology* bir çok-etmenli sistem üyesi için herhangi bir bilgi toplama ve muhakeme (reasoning) kaynağını temsil eder. Ontolojiler, belirli bir iş alanı içeriğini sağlayan, çok-etmenli sisteme ait bilgi tabanını oluşturmaktadır.

Ontology üst-varlığının *Organization* Ontology, *Service Ontology* ve *Role Ontology* adı verilen alt-varlıkları ilgili üst-model varlıkları tarafından kullanılmaktadır. Örneğin servislerin anlamsal arayüzleri ve yetenek tanımları *Service Ontology*'e bağlı olarak oluşturulmaktadır ve bu ontoloji *Semantic Web Agent*'ları tarafından *Semantic Web Service*'lerinin bulunması ve çalıştırılması için kullanılmaktadır.

4. Anlamsal Web Tabanlı Etmen Sistemleri için Platform Bağımsız Model Önerisi

Anlamsal web tabanlı etmen sistemleri için önerdiğimiz [11] üst model, sistemde yer alan genel kavramları ve yapıları içermektedir. Bu üst model, bir etmen sistemi için Model Tabanlı Mimari kapsamında platform bağımsız model ve platform bağımlı model yapılarını tanımlamak için gerekli olan anahtar noktaların tanımlanması için önemlidir. Ancak bu üst model, anlamsal web tabanlı etmen sistemleri için platform bağımsız model üst modeli olarak kabul edilemez. Bunun için üst modelin detaylandırılması gerekmektedir. Örneğin üst modelde, *SemanticWebService* ile *ServiceOntology* gibi yeni model elemanlarının etmen-servis etkileşimi doğrultusunda detaylandırılması gerekmektedir. Bu doğrultuda çözülmesi gereken sorunlardan ilki bu etmen-servis iletişiminde gerekli olan yapıların üst model içerisinde tanımlanması olacaktır.

Anlamsal web etmenlerinin bulunması ve çalıştırılması için bir anlamsal web servisi, servis arayüzünü (service interface) ve servis süreç mekanizmasını (service process mechanism) içermektedir. Anlamsal web etmen kaynak kodunu platform bağımlı model üzerinden üretebilmek için bu arayüz ve süreç yapılarının üst model içerisinde tanımlanması gerekmektedir. *OWL-S (Ontology Web Language-Service)* [22] ve *WSMO (Web Service Modeling Ontology)* [27] kapsamında web servislerinin anlamsal olarak modellenmesi üzerine çalışılmasına rağmen anlamsal web ortamında çalışan bu tip web servislerinin herhangi bir platformdan bağımsız olarak gösterilmesine ilişkin bir standart geliştirilmemiştir.

Gronmo [23] tarafından gerçekleştirilen çalışma, anlamsal web servislerinin Model Tabanlı Mimari

kullanılarak geliştirilmesi için iyi bir örnektir. Gronmo [23], çalışmasında platformdan bağımsız olarak anlamsal web servislerinin tanımlanabilmesi için bir UML profili tanımlamıştır. Bu UML profili sayesinde anlamsal web servislerini tanımlayan değişik platformlara ait OWL-S ve WSML dokümanlarını otomatik olarak üretmek mümkün olmaktadır. Bu çalışma anlamsal web servislerini model tabanlı geliştirmek için iyi bir örnek olmasına rağmen, anlamsal web servisleri ile etmenlerin etkileşimini içeren herhangi bir yapı bulundurmamaktadır.

Anlamsal web tabanlı etmen üst modeli için geliştirilmesi gereken diğer bir bölüm ise planlama alt yapısıdır. Davranış kütüphanesi içerisinde yer alan yeniden kullanılabilir plan alt yapılarından birisi de *Hierachical Task Network planning* adı verilen *HTN* planlarıdır [17]. Model Tabanlı Mimari bakış açısıyla HTN gibi planlama mekanizmaları tarafından tanımlanan plan yapıları platform bağımlı model seviyesinde yer almaktadır. Örneğin anlamsal web tabanlı etmen platformu olan SEAGENT içerisindeki plan yapıları, [14] ve [19]'da belirtilen plan mekanizmasına göre gerçekleştirilmiştir. SEAGENT platformunun üst modelini platform bağımlı model üst modeli olarak düşünürsek, HTN mekanizmasına özel yapılar bu üst model içerisinde yer alacaktır. Platform bağımsız model seviyesinde ise HTN ve diğer planlama mekanizmalarından bağımsız olarak genel planlama kavramlarıyla ilgili yapılar üst model içerisinde yer almalıdır.

5. Sonuç ve Gelecekteki Çalışmalar

Bu bildiride, anlamsal web tabanlı etmen yazılım sistemlerinin tasarımı ve geliştirilmesine yönelik model tabanlı bir yaklaşım tartışılmıştır. Literatürde bulunan etmen sistemlerinin geliştirilmesine yönelik modelleme dillerinin ve üst modellerin anlamsal web tabanlı etmen sistemleri için herhangi bir destek sağlamadıkları görülmüştür.

Model Tabanlı Mimari bağlamında Anlamsal Web yetenekli etmen varlıklarının bir etmen üst modelinde nasıl yer aldığı bu bildiride incelenmiştir. Bu tip çok-etmenli sistemlerin geliştirilmesi için Model Tabanlı Mimari kapsamında platform bağımlı ve platform bağımsız modellerin sahip olması gereken yapılar üzerinde durulmuştur.

İleriye yönelik çalışmalardan ilki, üst modelin gereksinimlere uygun üst varlıklarla genişletilmesidir. Statik bakış açısı yanında modelin üst yapılar arası etkileşimlerinin de ortaya konulması ikinci planlanan çalışma olarak tanımlanmıştır. Tüm bu alt yapının tamamlanmasından sonra alt yapıya uygun etmen üst modelleri arasında dönüşümlerin tanımlanarak SEAGENT [21] etmen geliştirme platformuna uygun kaynak kod üretilmesi hedeflenmektedir.

6. Kaynaklar

[1] Agrawal, A., Karsai, G., Shi., F. Interpreter Writing using Graph Transformations, *Vanderbilt University Technical Report*, ISIS-03-401, 2003.

[2] Kalnins, A., Barzdins, J., Celms, E. Model Transformation Language MOLA, Lecture Notes in Computer Science (LNCS), 3599/2005, MDAFA 2004.

[3] Molesini, A., Denti, E., Omicini, A. MAS Meta-models on Test: UML vs. OPM in the SODA Case Study, *CEEMAS 2005*, Budapest, Hungary.

[4] Perini, A., Susi, A. Automating Model Transformations in Agent-Oriented Modeling, *Agent Oriented Software Engineering Workshop (AOSE 2005)*, Utrecht, the Netherlands, 2005.

[5] Bauer, B., Odell, J. UML 2.0 and Agents: How to Build Agent-based Systems with the New UML Standard, *Journal of Engineering Applications of Artificial Intelligence*, 18(2):141– 157, 2005.

[6] Bernon, C., Cossentino, M., Gleizes, M., Turci, P., Zambonelli, F. A Study of some Multi-Agent Meta-Models, *Agent Oriented Software Engineering Workshop (AOSE 2004)*, New York, USA

[7] Jouault, F., Kurtev, I. Model Transformations with ATL, *MoDELS Satellite Events 2005*, Revised Selected Papers in LNCS 3844 Springer 2006, Montego Bay, Jamaica, 2005.

[8] Jayatilleke, G.B., Padgham, L., Winikoff, M. Towards a Component based Development Framework for Agents, *Second German Conference on Multiagent System Technologies (MATES 2004)*, Erfurt, Germany, 2004.

[9] Jayatilleke, G.B., Padgham, L., Winikoff, M. A Model Driven Development Toolkit for Domain Experts to Modify Agent Based Systems, *Agent Oriented Software Engineering Workshop (AOSE 2006)*, Hakodate, Japan.

[10] Benguria, G., Larrucea, X., Elvesaeter, B., Neple, T., Beardsmore, A., Winchester, M. A Platform Independent Model for Service Oriented Architectures, *I-ESA 2006*, Bordeaux, France, 2006.

[11] Kardas, G., Goknil, A., Dikenelli, O., Topaloglu, N.Y. Metamodeling of Semantic Web Enabled Multiagent Systems, *Multiagent Systems and Software Architecture (MASSA 2006)*, Erfurt, Germany, 2006.

[12] Bezivin, J. Model Driven Engineering: Principles, Scope, Deployment and Applicability, *Proceedings of the*

Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE'05), Braga, Portugal, 2005.

[13] Ferber, J., Gutknecht, O. A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems, *In: Proc. 3rd International Conference on Multi-Agent Systems*, IEEE Computer Society, 128-135, 1998

[14] Graham, J.R., Decker, K., Mersic, M. DECAF – a flexible multi agent system architecture, Autonomous Agents and Multi-Agent Systems, 2003.

[15] Odell, J., Nodine, M., Levy, R. A Metamodel for Agents, Roles and Groups, In: Agent Oriented Software Engineering V: 5th International Workshop, AOSE 2004. Lecture Notes in Computer Science, Vol. 3382, pp. 78-92. Springer-Verlag, Berlin Heidelberg (2005)

[16] Pavon, J., Gomez, J., Fuentes, R. Model Driven Development of Multi-Agent Systems, *Lecture Notes in Computer Science (LNCS)*, 4066/2006, EC-MDA 2006.

[17] Erol, K., Hendler, J.A., Nau, D.S. Complexity Results in HTN Planning, Ann. Math. Artif. Intell., 1996.

[18] Duddy, K., Gerber, A., Lawley, M. Model Transformation: A declarative, reusable patterns approach, *Proceedings of the 7th International Enterprise Distributed Object Computing Conference (EDOC'03)*, Brisbane, Australia, 2003.

[19] Sycara, K., Williamson, M., Decker, K. Unified Information and Control Workflow in Hierarchical Task Networks, *In: Working Notes of the AAAI-96 workshop 'Theories of Action, Planning, and Control'*, 1996.

[20] Object Management Group (OMG). MDA Guide Version 1.0.1. Document Number : omg/2003-06-01, 2003.

[21] Dikenelli, O., Erdur, R.C., Kardas, G., Gümüs, Ö, Seylan, I., Gurcan, Ö, Tiryaki A.M., Ekinci, E.E. Developing Multi Agent Systems on Semantic Web Environment using SEAGENT Platform, *Proceedings of the 6th International Workshop Engineering Societies in the Agents World (ESAW'05)*, Kusadasi, Turkey, 2005.

[22] The OWL Services Coalition. Semantic Markup for Web Services (OWL-S). http://www.daml.org/services/owl-s/1.1/, 2004.

[23] Gronmo, R., Jaeger, M.C., Hoff, H. Transformations between UML and OWL-S, *Proceedings of the 1st European Conferenc on Model Driven Architecture – Foundations and Applications (ECMDA-FA'05)*, Nuremberg, Germany, 2005.

[24] Depke, R., Heckel, R., Küster, J.M. Agent-Oriented Modeling with Graph Transformations, *Agent Oriented Software Engineering Workshop (AOSE 2000)*, Limerick, Ireland, 2000.

[25] Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web. *Scientific American*, 284(5):34– 43, 2001.

[26] Guessoum, Z., Thiefaine, A., Perrot, J., Blain, G. META-DIMA: a Model-Driven Architecture for Multi-Agent Systems, URL: http://www-poleia.lip6.fr/%7Eguessoum/MetaDima.html, (last accessed: 2006)

[27] WSMO Working Group. Web Service Modeling Ontology (WSMO). http://www.wsmo.org/.

Endüstri Oturumu

ASELSAN'da Yazılım Mimarileri Uygulamaları

Levent Alkışlar
ASELSAN A.Ş
alkislar@mst.aselsan.com.tr

Öz

Günümüzde yazılımlar, üretilen sistemlere kazandırdığı farklı yetenekler sayesinde ürün çeşitliliğinin artmasını ve ürünlerin müşteri taleplerine göre kişiselleştirilmesini sağlamaktadır. Artan ürün ve müşteri çeşitliliği, kısalan geliştirme süreleri ile yazılım geliştirme maliyetlerinin aşağı çekilmesi talepleri, kaynakların etkin kullanılması ihtiyacına, dolayısıyla yazılımların sistematik olarak yeniden kullanılması fikrinin doğmasına neden olmuştur.

Yazılımların yeniden kullanılabilirliğine etki eden faktörlerin başında, yazılım mimarileri, yazılım geliştirme organizasyonu ve süreçler gelmektedir. Kuruluşların başarılı şekilde yeniden kullanılabilir yazılımlar ve iş ürünleri üretebilmesi için bu üç unsurun da uygun şekilde yapılandırılması gereklidir.

Bu bildiride ASELSAN Mikrodalga ve Sistem Teknolojileri (MST) Grubu'nda yazılım yeniden kullanılabilirliğini sağlamak amacıyla uygulanmaya başlanan Yazılım Referans Mimarisi tanımlama ve bu mimari ile uyumlu bileşenlerin geliştirilmesi faaliyetleri ile Yazılım Ürün Hattı yaklaşımı ile uyumlu organizasyon önerileri özetlenecektir.

1. Giriş

Yazılım ürünlerinin yeniden kullanılabilirliğinin sistematik olarak sağlanması için 90'ların ilk yarısından sonra büyük boyutlu araştırma faaliyetleri yürütülmüş, bu çalışmalar günümüzde Yazılım Aileleri veya Yazılım Ürün Hattı olarak isimlendirilmeye başlamıştır. Bu yeni kavramlar öncelikle tüketim sektöründe faaliyet gösteren firmalar tarafından sahiplenilmiş, ancak günümüzde savunma sektöründeki firmalar tarafından da uygulanmaya başlamıştır [1], [2], [3], [4].

ASELSAN artan ürün çeşitliliği, sözleşme sayısı, ihracat olasılıkları ve altyüklenici kullanım ihtiyacı ile azalan geliştirme süreleri karşısında daha hızlı, daha iyi ve daha ucuz ürünler çıkartabilmek maksadıyla dünyadaki gelişmelere paralel olarak, 2000'li yılların başından itibaren yazılım yeniden kullanılabilirliğinin sistematik olarak sağlanmasına önem vermiştir.

2. ASELSAN

ASELSAN faaliyet sahalarına göre, Haberleşme Cihazları (HC), Mikrodalga ve Sistem Teknolojileri (MST) ile Mikroelektronik, Güdüm ve Elektro-Optik (MGEO) Grubu olmak üzere üç grup halinde organize olmuştur. ASELSAN MST Grubu faaliyet sahasına, Radar Sistemleri, Elektronik Harp Sistemleri, Savunma ve Silah Sistemleri, Ateş Destek Otomasyon Sistemleri, Komuta Kontrol ve İstihbarat Sistemleri ve Deniz Savaş Sistemleri girmektedir.

MST Grubu, 1991 yılında geçirdiği organizasyon değişikliği ertesinde matris organizasyonu benimsemiş, proje yönetim ve tasarım bölümleri oluşturulmuş, 1995 yılında ilk defa bir Yazılım Mühendisliği bölümü kurularak değişik tasarım bölümlerinde yürütülen yazılım geliştirme faaliyetleri tek bölüm sorumluluğu antında toplanmıştır.

Yine yazılım geliştirme ve yönetim süreçlerinin geçmişi 90'lı yılların başına kadar gitmekte, ilk süreç ekiplerinden biri olan Yazılım İyileştirme Süreç Ekibi yaklaşık 15 senedir faaliyet göstermektedir.

İlgilenilmekte olan faaliyet alanlarının çeşitliliği dolayısıyla sistemler çok değişik türde ve teknolojide yazılım ürünü barındırmakta, bunlara yönelik zengin bir yazılım geliştirme altyapısı idame edilmektedir.

3. Yapılanlar ve tecrübeler

2000'li yıllardan sonra başlatılan çalışmalar ile öncelikli olarak faaliyet alanlarında kullanılacak ortak bir Yazılım Referans Mimarisi çıkartılması hedeflenmiştir. Daha sonra bu mimariye uygun ilk yeniden kullanılabilir ürün örnekleri oluşturularak ardışık projelerde kullanılması sağlanmıştır. Ardından gelen dönemde, daha çok başlangıçta yürütülen faaliyetlerin yönetsel ve kurumsal eksiklerinin

giderilmesi yönünde çalışılmış, organizasyon ve süreçler üzerindeki etkileri incelenmiştir. Bundan sonraki dönemde yeniden kullanılabilirliğin süreçlere yansıtılması yönünde çalışılmaya devam edilecektir [5].

Bu çalışmalar sonrasında ortaya çıkan kazanımların bazıları aşağıda verilmiştir:

- Ortak kullanılacak bir referansa, ortak paylaşılan bir dil ortaya çıkmıştır.
- Projelerde yeniden kullanılabilecek ortak ve farklı yapıtaşlarının görülmesi yardımıyla planlamanın daha etkin yapılması sağlanmıştır.
- Ana faaliyet sahamıza giren ve dışında kalan, dolayısıyla alt yüklenicilere verilebilecek işler ayrıştırılabilmiştir.
- Kaynaklarımızı odaklayacak ana faaliyet noktalarının görülmesi sağlanmıştır.

4. Gelecek için planlar

Yapılmakta olan çalışmaların devamı niteliğinde planlanan çalışmalar ise aşağıda özetlenmektedir:

- Mimariyi oluşturan bileşenlerin gerçeklenmesinin tamamlanması ve kullanımının artırılması,
- -bilirlik analizlerini yaparak genel mimari ve değişik yazılımları için oluşturulan özel mimari içinde yer almasının sağlanması,
- Hazırlanan referans mimarinin, projelerdeki ve teknolojideki gelişmelere uygun olarak yeniden yapılandırma yoluyla yaşatılması,
- Mimari bileşenlerin geliştirildiği pilot projelerde kazanılan tecrübenin, organizasyon ve süreçlere yansıtarak, kurumsallaşmanın kalıcı hale getirilmesi,
- Sistem yaşam döngüsünün başında ve sonunda yer alan İş Geliştirme ve Bakım dönemindeki faliyetlere vurgunun artırılması,
- Yatay ve dikey iletişimi artırmak amaçlı kalıcı yöntemler geliştirilerek uygulamaya alınması.

5. Sonuç

ASELSAN dünyadaki gelişmelere paralel olarak, yazılımların yeniden kullanılabilirliği ve bunun kurumsallaşma yoluyla sistematik olarak gerçekleştirilmesi yönündeki çalışmalarını sürdürmektedir.

Bu çalışmalar içerisinde bir Referans Mimari oluşturulmuş, mimariyi oluşturan bileşenler geliştirilmiş, pilot projeler yoluyla organizasyona ve süreçlere yönelik etkiler incelenmiştir.

6. Kaynaklar

[1] Len Bass, Paul Clements, Rick Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.

[2] Jan Bosch, *Design & Use of Software Architectures – Adopting and Evolving a Product-line Approach*, Addison-Wesley, 2000.

[3] Paul Clements, Linda Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002.

[4] Daniel J. Paulish, *Architecture-Centric Software Project Management – A Practical Guide*, Addison-Wesley, 2002.

[5] ASELSAN MST Grubu, *REFoRM – RADAR Elektronik Harp Fonksiyonel Referans Modeli*, 2004.

Impact of Architectural Concerns on Continual Achievement of Enterprise Applications

Semih Çetin
Cybersoft Information Technologies
semih.cetin@cs.com.tr

Abstract

Enterprise applications are mission critical for organizations. Though enterprises are using systematic or ad-hoc methods for architectural design, currently there are several initiatives that see enterprise application architecture as their natural playground, like Model-Driven Architecture and Service-Oriented Architecture. Meanwhile, emerging needs of business urge the use of certain application and/or component frameworks as well as the separation of particular concerns such as business processes and business rules. Composing the designs for functional and non-functional requirements is another issue having a major impact on the success of enterprise applications, too. Now is the time to investigate how these approaches can provide added value to the architectural modeling of practical solutions that enterprises are in search of. This paper briefly summarizes the architectural modeling techniques used in industrial applications and outlines the recent trends in state-of-the-art approaches for architectural design. Readers may find interesting results in the examples taken from the own experiences of Cybersoft.

1. Introduction

Applications, operating systems, database systems, hardware architecture and administration concepts must be orchestrated in industrial applications to yield an optimized system architecture that tackles performance, stability, security, maintainability, and eventually total cost of ownership. In practice, it is always a holistic view that is needed. In this paper, the practical underpinnings of enterprise application architectures will be explained together with how things have changed in the architectural design of industrial applications during the last decade. While doing that Cybersoft experiences will be summarized as well to present more practical concerns to the readers.

Enterprises today operate quite differently than they had been doing a decade ago. They are faster and more efficient, 24/7 accessible and mostly providing self-service functionality. Web and related technologies are ruling the ways enterprises are making business. The story has started with core issues like the use of centralized processes, e-mail in the provision of information, on-line community facilities, and today it has reached to the level of excellence with Business-to-Business (B2B) communication, Web services, process orchestrations and even business process choreography.

All these "the dream comes true" approaches have been realized by the consequences of challenging technologies that may be comprised of pre-built, tested and proven third party components. Even not less than that are the infrastructures and frameworks to keep all these components working like a Swiss watch. Apart from these tangible tools that we have been using, the underlying architectural designs play also a major role in the success of enterprise applications, which lead to certain quality attributes like performance, robustness, reliability, portability, etc. achievable today.

The quality attributes of enterprise software systems are principally determined by the system's software architecture, not merely by the software process life cycle. That is, in enterprise systems, the achievement of qualities depends more on the overall software architecture in terms of application partitioning, tier design, framework selection, and service integration than on code-level practices such as methodologies, detailed design, algorithms, data structures, testing, and so forth. This is not to say that choice of functionality issues is less important to achieve the software quality attributes, but rather that they are complementing the structuring of enterprise systems.

The paper starts with running an eye over the major challenges of enterprise applications development. Then comes the disclosure of essential architectural environment within the context of functional and non-functional expectations driven by the stakeholders'

sight. After that, it portrays the roadmap for successive achievement of enterprise applications and exposes the cases that have already been implemented accordingly. Finally, further directions will be given and the paper will end with the conclusion.

2. Challenges for enterprise applications

Enterprise applications are usually business-critical software applications with many users in large organizations. Those enterprise applications typically process large volumes of complex, persistent data in business processes in line with specific business rules. Furthermore, they need to enable simultaneous access by various users inside and outside the organization, pose demanding security requirements, and must coalesce other applications or interoperate with them.

Essentially, Enterprise Application Development (EAD) is a multi-disciplinary process covering not only Information and Communication Technologies (ICT), but also several issues like policy determination, social-cultural aspects and so forth [20]. Thus, EAD faces quite a lot of challenges ranging from the user interfaces to the use of emerging technologies like Semantic Web. However, this paper merely sticks into the ICT perspective here to reveal the main driving factors and principal motivation to portray for sustainable achievement of enterprise applications. In that sense, the following major challenges can be listed.

2.1. High volume of transactions

Non-uniform distribution of service requests at peak times geometrically complicates the design of server farms. Cybersoft has witnessed this concern in several intranet and internet-based implementations among which Electronic Taxation System of Turkey (VEDOP) is the typical one with peak time utilization depicted in Figure 1 and 2 within hourly demands in percentages.

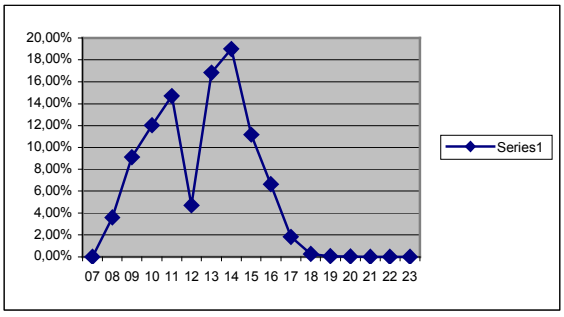


Figure 1. Hourly demands in intranet-based eTaxation application

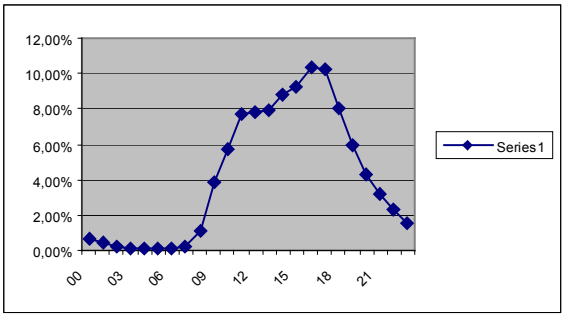


Figure 2. Hourly demands in internet-based eDeclaration application

2.2. Coherent and flexible infrastructure

Many enterprise applications include e-Services to be sufficiently served on the Web. Such services cannot be provided without having a proficient backend intranet solution that leans on the same or very similar architectural philosophy or seamless integration with the backend transactions. In any case, enterprises need a powerful infrastructure for loosely coupled service orientation [3, 4].

2.3. Sizing

Being one of the most challenging issues, the accurate sizing of enterprise applications prerequisites coherent/flexible architectures for modular services and business processes. Understanding a business domain, its both functional and non-functional requirements, identifying the needs of different stakeholders are all significant issues for the correct sizing of enterprise applications that will form the baseline for software architecture as well.

2.4. Interoperability

Providing the composite services requires intricate interoperability of relations mentioned in Section 3.1 with many others. Interoperability is the distinguished quality attribute of enterprise applications since the needs of enterprise business urge to have collaboration among different applications whose vendors may vary in names and numbers. The protocols, standards, data and control flow are all major concerns to provide the interoperability of applications [14].

2.5. Security

Enterprise services especially pertaining to financial transactions should be secured at every possible layer.

We all know that in Cybersoft most by heart from the experience of VEDOP [12] as the largest public sector implementation of Turkey in many senses. Moreover, by providing the core electronic transaction-processing infrastructure to Central Registry Agency (Merkezi Kayıt Kuruluşu – http://www.mkk.com.tr), we know that financial applications need a very special care for security in architectural design [7].

2.6. Identity management

If the number and variety of stakeholders accessing the system increase then registration, authentication and authorization will become quite complicated. Enterprise applications mainly need restricting the user profiles to access the security sensitive data as they are or delegated to be and within the context of granted permission levels. When the application is deployed onto server farms, the management of "single sign-on" procedures, synchronization of access rights and the administration of audit-trails become real issues.

2.7. Change management and dynamism

The dynamism of business processes is a core issue today for enterprise applications. The question is how to solve that issue by means of agile processes [1, 25] or reflective infrastructures [8]. We know that the set of change requests both on internal and external enterprise services affects the way we structure these services accordingly, and further enforces dynamic [8] as well as reflective software architectures [22].

2.8. Offline and batch services

The transformation of enterprise applications from merely the end user-based interactive ones to the whole set of stakeholders-based ones reveals the inevitability of offline and batch services particularly for the enterprise-to-enterprise streaming. B2B communication channels create vast amount of transaction loads on enterprise applications. Clustering of data provisioning services plays a major role on the performance and scalability of such applications.

2.9. Continuance of quality

Every new service added to enterprise applications should have a certain level of quality as claimed by the previous ones. However, it is really hard to achieve that unless one distills these quality factors into predefined architectural concerns by setting the infrastructure for orchestration/choreography of these services [5].

3. Environment for enterprise applications

Though the term "environment" is a multi-faceted definition, it may be thought of a generic context by which surrounding conditions and parameters are clearly specified according to the prerequisites of a system. Similarly, there may be different definitions for the "Enterprise Application Environment", however the ideal definition should reveal functional expectations such as policies, strategies, legal, cultural and social aspects together with non-functional expectations like flexibility, consistency, reliability, accessibility, and the overall quality of service driven by the stakeholders' sight. Undoubtedly, then, the stakeholders of enterprise applications should be understood first. This approach is quite similar to the "perspectives" proposed by the Kruchten's "4+1 View" approach [17], however it is more stakeholder-oriented.

3.1. Stakeholders

Today, EAD technologies fundamentally alter the way people and organizations conduct their daily lives/operations as well as revolutionize the "customer-to-business (C2B)", "customer-to-agency (C2A)", "business-to-business (B2B)", "business-to-agency (B2A)", and "agency-to-agency (A2A)" interactions. Besides that enterprises should reorganize themselves internally just to cope with the complexity of governing such a vital network of relations. Hence, they must harness existing and emerging eTechnologies to keep all stakeholders aligned [23].

From the quality of services, stakeholders of enterprise applications can be divided into two main categories of "external/internal" and each category has items that can be matched with aforementioned relations to create the dependency matrix given in Table 1.

Table 1. Stakeholder relations in EAD

Stakeholders	C2B	C2A	B2B	B2A	A2A
External					
Customers	✓	✓			
Enterprises			✓	✓	
Service Providers	✓	✓	✓	✓	✓
Application Providers	✓	✓	✓	✓	✓
Governing Agencies	✓	✓	✓	✓	✓
Internal					
Executives	✓	✓	✓	✓	✓
End Users		✓		✓	✓
Administrators		✓		✓	✓
ICT Departments		✓		✓	✓
Operations Departments		✓		✓	✓

The stakeholders of enterprise applications have not been taken into account sufficiently so far, especially from the service quality point of view. As a result, merely the customers are focused and the services are designed accordingly. However, the wide variety of stakeholders mentioned in Table 1 is expected to involve in enterprise applications, thus the essential EAD environment should rise on the shoulders of several factors depicted in Figure 3 as consecutively stand on common standards, principles and the core software architecture.

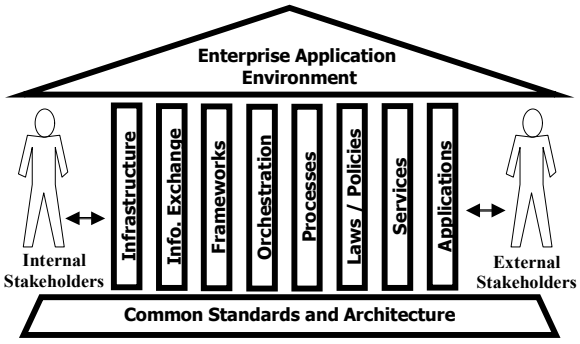


Figure 3. Environment for enterprise applications

This setup matches the relations and activities of internal and external stakeholders through a well-defined infrastructure providing proper information exchange facilities that run on both application and component frameworks. On these frameworks, inner processes are managed and orchestrated with outer processes under the supremacy of internal policies and external laws, which are implemented as applications providing independent services to inside and outside of the enterprise application implementation.

3.2. Functional expectations

Every implementation of an enterprise application is unique. So, when such projects are planned, there has been a little choice but to opt for custom development. This can be costly and time-consuming. However, every enterprise application shares something similar to provide a service and therefore have similar technical requirements. In the current environment, any specific know-how obtained during a project stays with that project, so the next implementation cannot benefit.

Off-the-shelf packages may have some of this required functionality, but even they cannot be used to easily put forward enterprise applications without expensive and risky customization. Hence, whatever needed today is an intelligent framework for enterprise applications that gives you the best of both worlds by merging the flexibility of custom functionality with the stability and cost-effectiveness of using off-the-shelf services. This seems almost impossible but surprisingly achievable at least in parts by proper structuring of software components around stable infrastructures [21].

3.3 Non-functional expectations

There are mainly two approaches to be followed for ensuring the software quality; one is the assurance of development processes; so-called "production quality", and the other is the whole set of quality factors the end product should have; a.k.a. "product quality". Both are important, but what we have mostly experienced in Cybersoft is that "sustainable" quality of enterprise applications can be better achieved by emphasizing the product quality and structuring the software processes around it. Revisiting the major challenges for enterprise applications given in Section 2 reveals the same, too.

The principles of software architecture also suggest this way by trying to embroider the quality factors into software components methodically instead of always expecting the duly craftsmanship of individuals. The ISO/IEC 9126 classifies these quality factors into 6 major categories [15] to systematically manage and correlate them with the stakeholders. Such a correlation between software quality factors and the stakeholders of enterprise applications may end up with a matrix shown in Table 2.

Table 2. Stakeholders and quality factors

Stakeholders	Q1	Q2	Q3	Q4	Q5	Q6
External						
Customers	✓	✓	✓	✓		
Enterprises	✓	✓	✓	✓		✓
Service Providers	✓	✓		✓	✓	
Application Providers	✓	✓		✓	✓	✓
Governing Agencies	✓	✓	✓	✓		✓
Internal						
Executives	✓	✓	✓	✓	✓	✓
End Users	✓	✓	✓	✓		✓
Administrators	✓					
ICT Departments	✓	✓	✓	✓	✓	✓
Operations Departments	✓	✓		✓	✓	✓

Q1: Functionality **Q2**: Reliability **Q3**: Usability
Q4: Efficiency **Q5**: Maintainability **Q6**: Portability

4. A roadmap for sustainable applications

There is a paradox for enterprise applications: every one of them is different, yet many of them share some commonalities that can be modeled straightforwardly if

the basic principles of software architecture are applied rigorously. On top of that, the relevant issues such as separation of concerns, component and application frameworks, and even "Software Product Lines (SPL)" approach take the reusability base to the extremity [24].

4.1 The architecture

The software architecture of a computing system is the structure or structures of the system, which combine software components, the properties of those components, and the relationships among them [6]. The definition originally exposes the identification of repeatable parts in a software system that can be used later to minimize the risk. Hence, software architecture provides a hallmark for logically assuring the software quality factors also known as non-functional issues, which can be physically achieved then by means of application/component frameworks [2, 18].

Architecture of software systems cannot be conceptualized from a single point of view and hence Kruchten has suggested four identified views: logical, development, process and physical, which are all segmenting the quality factors into a software application through the scenarios from the viewpoint of stakeholders [17]. Thus, enterprise applications are expected to demand on the theory and best practices of software architecture principles as well. At the end, these principles target the proper partitioning of applications in terms of reusable assets with respect to the architectural aspects of different enterprise applications. Supportively, there may be a common architectural model for enterprise applications at the meta-level that can be used successively, and such a meta-model can be constituted by means of the common quality factors prioritized by the stakeholders of such applications.

4.2 The architectural modeling approach

Software architecture has gained a vast impetus for improving the quality of software applications; hence architectural modeling has become an area of intense research [11]. An extensive study has been carried out to reveal the evolution of software architectures in line with achieving software quality [24]. Supportively, the quality attributes of software have been classified [15, 18], architectural concerns have been identified with respect to conceptual [3, 14] and semantic [2, 21] models together with viewpoint and stakeholder-based reasoning [6].

Regarding architectural modeling, we (as Cybersoft) agree with [10]: "a future work is required to develop systematic ways of bridging quality requirements of software systems to the architecture; unsolved problem is how to take architectural concepts better to analyze software systems for quality attributes in a systematic way". The Cybersoft approach summarized below tries to solve this issue as explained and exemplified in [7].

Managing the concerns in architectural modeling is not easy particularly for identifying the risks, tradeoffs and sensitivity points. Two things are vital for concern management: localization of architectural concerns and representation of these concerns in a specific solution. Similar to other modeling approaches, this perspective divides the software architectural modeling process into two: "identification of the problem domain" and "description of the solution domain".

Software architecture modeling is quite complicated in that sense since both of the domains are not trivial to identify. There are no standard ways to associate them, either. We introduce a modeling approach to localize architectural concerns in multiple concern spaces for both domains, and symmetrically align them as shown in Figure 4. While our approach separates the modeling of problem and solution domains in two distinct steps, it does not imply that they are not parallel activities or not intertwined with each other as addressed by Hall et al in the "Twin Peaks" model [13]. In fact, the mapping process in Step 4 may result with returning back to Step 2 and customize the quality attributes.

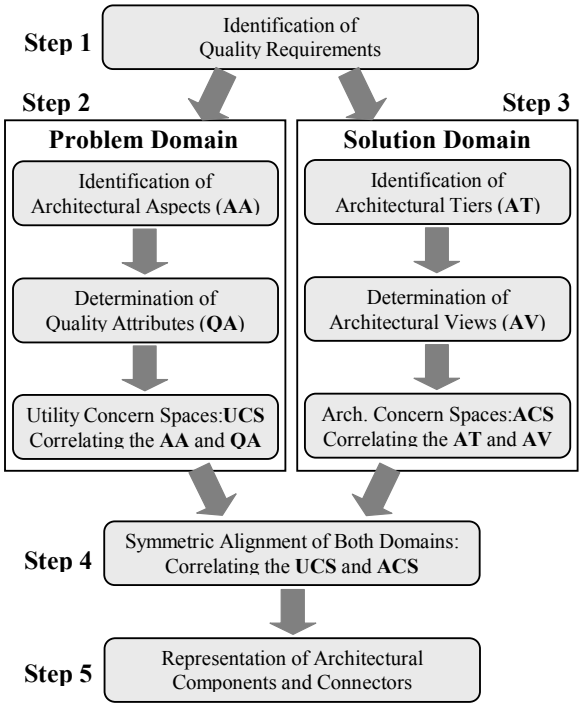


Figure 4. Architecture modeling approach

4.2.1. Identifying quality requirements. Starting point of this approach is the identification of quality requirements, which may vary from one application to another. In order to collect the actual set of quality requirements, different methods can be used and we shall not propose another one here since this is beyond the scope of our architectural modeling approach. However, what we have experienced is that scenario-based and stakeholder-oriented ones usually end up with more realistic and near to complete set of quality requirements than the others.

4.2.2. Identifying problem domain. Identified set of quality requirements will form the problem domain concerns in multiple concern spaces called as Utility Concern Spaces (UCS). UCS takes the name from ATAM since it uses a specific technique to map the architectural aspects to quality attributes first in a hierarchical decision tree called as "utility" tree.

In this approach, UCS is constructed by correlating the quality attributes taken from any quality model such as [12, 19], and the architectural aspects categorized by the architect in modeling process. The problem domain is modeled as a matrix presented in Figure 5 where the architectural aspects show the rows and derived quality attributes constitute the columns of this matrix.

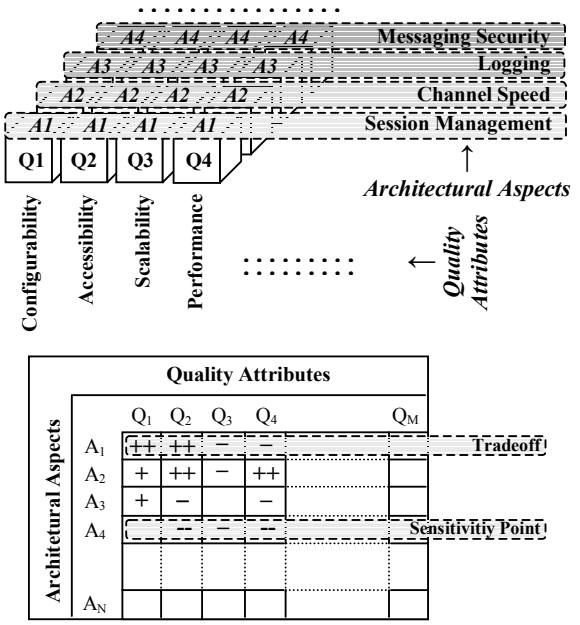


Figure 5. Utility concern spaces

UCS is an NxM matrix to be formed by the architect as shown in Figure 5. Each cell is shown like "UA_4Q_2" where "U" represents the Utility Concern Space, "A_4" denotes the fourth architectural aspect; Messaging Security in our example, and "Q_2" shows the second quality attribute; Accessibility in this case. The unique UCS cell identifiers will be used to form the symmetric alignment matrix later in Step 4.

The symbols like "-" or "++" in Figure 5 denote the "correlation coefficient", i.e. strength, of mappings between architectural aspects and quality attributes. The lack of any symbol means there is no correlation. The "+" or "-" symbol denotes a positive or negative correlation, and "++" or "--" symbol shows the stronger correlations. The coefficients play an essential role for identifying the "sensitivity points" - parameter to which some quality attribute is highly related - and "tradeoffs" - factor that affects many quality attributes in opposite directions [16]. One can easily identify the sensitivity points in a UCS matrix if we have only plus or minus signs in an individual row. However, coexistence of both plus and minus signs in a single row means a basic or strong tradeoff due to correlation coefficients.

4.2.3. Describing solution domain. Akin to problem domain concerns, solution domain concerns are formed into Architectural Concern Spaces (ACS), which is the correlation of architectural tiers and architectural views given by the matrix in Figure 6.

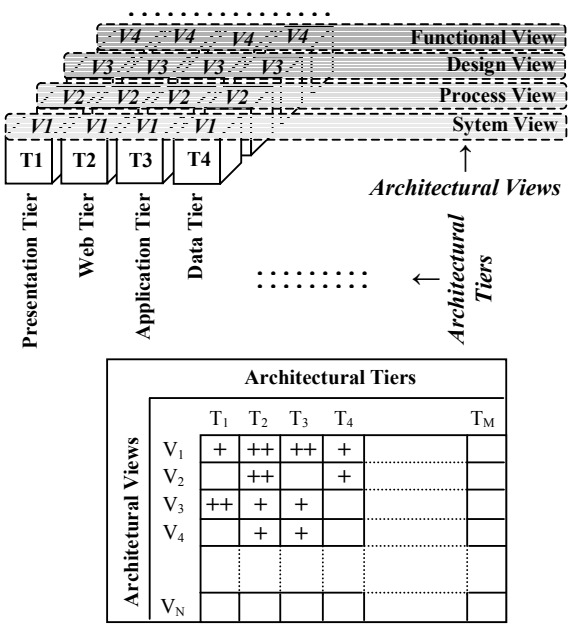


Figure 6. Architectural concern spaces

Architectural tiers are well-known tiers of "N-tier" model where different architectural styles can be used in a single tier or across the multiple tiers. Similarly,

architectural views are customized from Kruchten's "4+1 View" [17] as follows: Functional View shows functions, system abstractions, and domain elements as components; but dependencies as connectors. Design View shows services, components, and aspects as components; whereas invokes, calls, and joinpoints as connectors. Process View shows processes, threads, and workflows as components; but synchronization, control/data flows, and events as connectors. System View shows servers and networking as components; but protocols and message queues as connectors.

16 distinct concern spaces are shown in this matrix, but there may be more or less as the number of tiers and views differs. For instance, an architectural design may require a further "Business Processing Tier" or a separate "Development View" with respect to different quality requirements identified at Step 1. Similar to UCS, the concern spaces are shown as "AV_3T_1" where "A" represents the Architectural Concern Space, "V_3" denotes the third architectural view; Design View in this case, and "T_1" denotes the first architectural tier; Presentation Tier in the example. The unique ACS cell identifiers will be used again to form the symmetric alignment matrix later in Step 4. Figure 6 also includes the correlation coefficients like Figure 5 has, but note that ACS may have only positive correlations here.

4.2.4. Symmetric alignment of domains. Symmetric alignment is a join step after structuring the utility and architectural concern spaces separately. In fact, symmetric alignment is correlating the UCS and ACS in another matrix as shown in Figure 7.

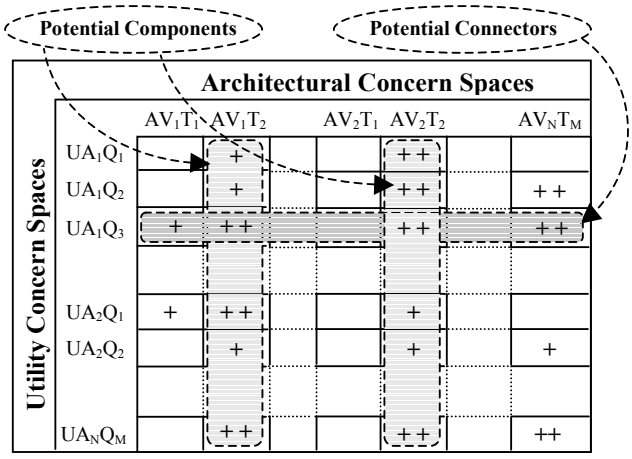


Figure 7. Symmetric alignment matrix

Symmetric alignment matrix has the rows formed by UCS cells given in Figure 5 and the columns formed by ACS cells given in Figure 6. Besides, every cell of this alignment matrix identifies the place (ACS) where a UCS should be related with solution elements. In fact, this association is not an exhaustive one-to-one pairing. The prioritization of UCS and ACS pairing is directed by the correlation coefficients of ACS matrix in a way that the ACS cells with strong correlation coefficients (having "++" symbols like AV_1T_2 in Figure 7) must be attempted first and then the others containing normal coefficients (having the "+" symbols like AV_3T_2). The cells without any correlation coefficient in ACS matrix should not be attempted for pairing with UCS cells. For example, the column AV_2T_1 in Figure 7 is left blank since the AV_2T_1 cell in Figure 6 does not contain any correlation coefficient.

The major contribution of this matrix is its guidance in identifying the architectural components, connectors and properties. Potential components can be realized by analyzing the columns of this symmetric alignment matrix. If a column contains many coefficients, then it means that several UCS are tangled in an ACS, and such concerns may be better abstracted with separate architectural components. For instance, AV_1T_2 column of Figure 7 signals a potential architectural component since this column has strong mappings to several UCS.

The row-wise distribution of correlation coefficients signals a potential architectural connector. In fact, such a distribution means that a single UCS has solution elements scattered through many ACS that may require a connection between architectural components. For example, Figure 7 signals a potential connector resulted by the scattering of UA_1Q_3 through different ACS, and it may probably bridge the architectural components identified under AV_1T_2 and AV_2T_2 columns.

Architectural properties can also be identified from the intersection of UCS and ACS. For example, the potential component under AV_1T_2 column of Figure 7 is expected to expose the properties like "clustering of secure Web servers" since this ACS is correlating the System View and Web Tier, and intersected by UA_1Q_3 row as a UCS that correlates the Session Management and Scalability.

4.2.5. Representing components and connectors. After identifying the components and connectors, an architectural model should represent them somehow. There are different Architecture Description Languages (ADLs) and other techniques devised for that.

4.3 The framework

Frameworks help architects/designers to partition their applications in a way that they can develop/maintain each part separately and integrate

easily. Thus, frameworks are quite helpful to achieve continual success in actual implementations but they are very limited to help modeling the reusability at early phases of design, stemming from the lack of "semantic" capabilities.

However, another field of software architectures has attacked this problem, thanks to the SPL concept. An SPL is a set of software-intensive systems sharing a common, managed set of features that satisfies the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. The key objectives of SPLs are to capitalize on commonality and manage variation thus reduce the time, effort, cost and complexity of creating and maintaining a different product line of software alike. The key for the effective resolution of these issues is the use of a product line architecture that allows an organization to identify and reuse software artifacts for the efficient creation of products sharing a set of commonalities, but varying in known and managed ways. The architecture, in a sense, is the glue that holds the product line together.

Having this vision of SPL approach and the significance of architectural aspects, we have started building an in-house SPL architecture 6 years ago at Cybersoft, which resulted the Web-based and platform independent Aurora framework [3, 4] that has been applied successfully to several enterprise applications from core banking to eGovernment. By customizing it a little bit for eGovernment development in Electronic Taxation System of Turkey, we realized that an ICT framework for eGovernment that is talented enough for the multi-tiered separation of architectural concerns from the functional expectations of an eGovernment solution has helped a lot in modeling the complicated services, labor division for development, ensuring the quality factors such as scalability, security as well as performance regardless of individual efforts, and managing tradeoffs between quality requirements systematically.

As a typical ICT framework, Aurora inherently enables the structuring and design of services starting from the presentation tier and ending with heterogeneous data sources. Even more than that, the very lightweight service architecture abstracts the sizzling complexities of several technologies like HTML and XML rendering, session replication for load balancing and fail-over management, deployment descriptors of service containers, and management of varying protocols for Enterprise Application Integration (EAI) from pure functional serviceability.

Aurora has a Service-Oriented Architecture (SOA) to be an enterprise service bus between different client requests of Rich Client Applications (RIA) as well as Web Services and data sources other than relational database management systems, as depicted in Figure 8.

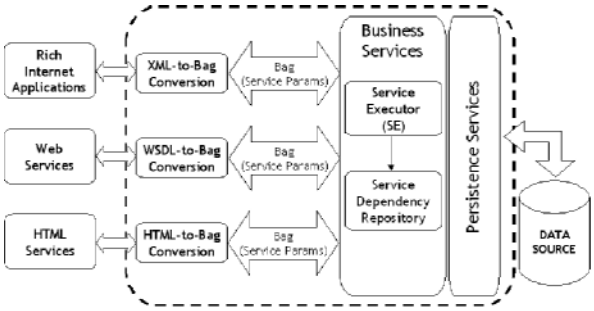


Figure 8. Service-Oriented Architecture of Aurora

4.4 Best practices in successive cases

Such an approach of clearly separating almost every architectural aspect from the functional concerns has some certain advantages. First, it hides the complexity issues from the actual developers, for instance Aurora has a supplementary drag-and-drop development studio for rendering the power screens (RIA screens) without requiring any single line of Graphical User Interface (GUI) code, thanks to its powerful XML rendering technology based on semantic User Interface Markup Language (UIML). This dramatically reduces the efforts for prototyping and simplifies the requirements elicitation in enterprise applications. Moreover, it provides a testing environment for "mock services" and "regression testing" by enabling test robots.

Second, service execution through the application tiers is totally abstracted away from the developers. The complicated issues such as protocol conversions, identity management through smart cards, and transaction management of heterogeneous data sources can all be buried into the architectural aspects, plugged into the framework and made available to developers via only known service development API of Aurora. For example, VEDOP has a complicated data tier so that a dedicated Data Access Layer (DAL) has been incorporated; however when conveying the same electronic taxation solution to Azerbaijan Ministry of Finance (AVIS) project has replaced DAL with Aurora's inherent object-to-relational mapping solution which did not affect the reuse of original VEDOP services on the backend. Similarly, the first eGovernment application using digital signatures in Turkey, Internal Processing Regime Project for Undersecretariat of Foreign Trade has introduced the

separate client-side JavaBean implemented by Aurora designers for that enriched the Aurora framework so that with the decision of Turkish Ministry of Finance, the electronic taxation application will be digital signature-enabled in a very short period of time and effortlessly.

Third, it simplifies the extension of enterprise applications. The beauty here in this approach is that changes and extensions to any enterprise solution are not directly applied in terms of functional development, but first put into an analysis of architectural concerns. As SPL suggests, the commonality/variation ratio is taken into account first, and then if it deserves, the change request is reflected to framework instead of individual enterprise implementations. Such an attitude has capitalized the enterprise know-how in terms of "reusable software assets" and several other applications can get benefit from that. In the digital signature case, other enterprises like Central Registry Agency of Turkey and Turkish Petroleum Organization have directly used that in their applications, which have already been developed on Aurora framework.

Fourth, surely it enables the use of engineering workforce in different enterprise projects. Transferring know-how from one enterprise implementation to another one is not easy. Sometimes forget about the know-how transfer, you cannot relocate the engineers. Maybe, the most notable part of this approach is that it accumulates the domain and ICT know-how mostly in the architectural aspects, rather in the minds of individuals or paperwork. Thus, the project managers can easily ask for the help of team members from other enterprise implementations and usually it is regardless of the application domain. Thanks to Aurora, we can easily transfer the engineering workforce among regional and even international implementations.

Nothing is "too good to be true" and this framework requires improvement as well. Although the complete architecture is managed by Aurora framework and partly self-expressive, it has no architectural modeling extensions that can be described by means of an ADL. Moreover, the framework is lack of semantic representation and a driving ontology, though the structure of the framework is quite suitable for that. Aurora can be used together with other testing robots but having an inherent test robot particularly for regression testing will certainly add a value. Finally, a supplementary tutorial and eLearning kit will enrich the environment.

In the future perspectives of Cybersoft, the Aurora framework will certainly play an important role in the way structuring enterprise applications in terms of domain specific kits for reusability [9].

5. Conclusions

Developing and maintaining enterprise applications are not easy. They compel visionary perspectives, never-ending dedication, and creative approaches for continual success. Having the scale of enterprise, they embody extreme difficulties inherently resulting from the high quality expectations of stakeholders in every sense. Being accessible to everyone also raises this difficulty in exponential orders, and makes the Web-based enterprise applications one of the hardest ICT applications, even if it is not the most complicated one today.

There are several proposals to overcome this intricacy: some people attack social concerns and legal issues, some others try to simplify the development processes, and yet others are looking for brand new technologies such as mobile phones and PDAs. This paper proposes maximizing the reuse of architectural aspects in enterprise applications for continual success rather than putting forward a brand new idea like "what will be the next big thing?". Throughout the years, we have been involved in several enterprise scale ICT projects and what they observed is simply nothing than the following: "reuse is indispensable at every stage of developing software".

The field of software architecture as well as relevant tools and techniques such as frameworks and SPLs put forward a systematic approach to reuse in software systems. By leveraging these concepts, we portrayed a practical roadmap to show how reuse of architectural aspects may help organizations achieve successive enterprise applications. The roadmap in our case highly demands on an SPL-based framework capitalizing the reusable software assets in a structured way so that they can be reused in its extremity for high quality solutions. Such a vision has brought the most reputable "IT award for eGovernment development" in the world to the company in the past and the we believe that this visionary perspective helped a lot for that.

6. References

[1] Z. Aktas, S. Cetin, "We Envisage The Next Big Thing: Knowledge Engineering", Integrated Design and Process Technology, San Diego – USA, 2006.

[2] R. J. Allen, "A Formal Approach to Software Architecture", Ph.D. Thesis, Carnegie Mellon University, School of Computer Science, CMU-CS-97-144, 1997.

[3] N. I. Altintas and S. Cetin, "Integrating a Software Product Line with Rule-Based Business Process Modeling",

Trends in Enterprise Application Architecture (2005), LNCS 3888, pp. 15-28, 2005.

[4] N. I. Altintas, M. Surav, O. Keskin and S. Cetin, "Aurora Software Product Line", 2nd National Software Engineering Symposium, Ankara-Turkey, 2005.

[5] N. I. Altintas, S. Cetin, and A. Dogru, "Industrializing Software Development: The "Factory Automation" Way", Trends in Enterprise Application Architecture (2006), Accepted Paper, 2006.

[6] L. Bass, P. Clements and R. Kazman, "Software Architecture in Practice", 2nd Ed., Addison Wesley, 2003.

[7] S. Cetin, N. I. Altintas, and C. Sener, "An Architectural Modeling Approach with Symmetric Alignment of Multiple Concern Spaces", ICSEA 2006 IEEE Conference, 2006.

[8] S. Cetin, N. I. Altintas, and R. Solmaz, "Business Rules Segregation for Dynamic Process Management with an Aspect-Oriented Framework", Business Process Management 2006, LNCS 4103, 2006.

[9] S. Cetin, N. I. Altintas, and O. Tufekci, "Improving Model Reuse with Domain Specific Kits", Proceedings of First International Workshop on Model Reuse Strategies (MoRSe), 2006.

[10] L. Dobrica and E. Niemela, "A Survey on Software Architecture Analysis Methods", IEEE Transactions on Software Engineering, Vol. 28, No. 7, 2002.

[11] D. Garlan, "Software Architecture: A Roadmap in The Future of Software Engineering", ACM Press, 2000.

[12] Y. Gogebakan, M. Kayrak, and M. Atug, "Virtual Revenue Administration, Turkey: An Innovative Implementation of ICT and BPR towards a Citizen-Centric Tax Authority", eChallenges, Barcelona, 2006.

[13] J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh, and L. Rappanotti, "Relating Software Requirements and Architectures using Problem Frames", 2002.

[14] IEEE, "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems", IEEE Std 1471.

[15] ISO/IEC 9126-1:2001, "Software Engineering — Product Quality", Part 1: Quality Model.

[16] R. Kazman et al, "The Architecture Tradeoff Analysis Method", 4th International Conference on Engineering of Complex Computer System, ICECCS, 1998.

[17] P. Kruchten, "Architectural Blueprints – The "4+1 View" Model of Software Architecture", IEEE Software, pp. 42-50, 1995.

[18] M. Larsson, "Predicting quality attributes in component-based software systems", Ph.D. Thesis, Dept. of Computer Science and Engineering, Mälardalen University, 2003.

[19] N. Lassing, D. Rijsenbrij, and H. Vliet, "Using UML in Architecture-Level Modifiability Analysis", ICSE Workshop on Describing Software Architecture with UML, pp. 41-46, 2001.

[20] K. C. Laudon and J. P. Laudon, "Essentials of Management Information Systems: Managing the Digital Firm", 6th Edition, Prentice Hall, 2004.

[21] OMG, "Model Driven Architecture", http://www.omg.org/mda.

[22] O. Onak and S. Cetin, "Reflective middlewAre for Manageable Business mOdels (RAMBO)", International Informatics Conference, Eskisehir – Turkey, 2005.

[23] M. Shaw and D. Garlan, "Software Architecture – Perspectives on an Engineering Discipline", Prentice-Hall, 1996.

[24] M. Svahnberg, "Supporting Software Architecture Evolution", Ph.D. Thesis, Blekinge Institute of Tech., 2003.

[25] O. Tufekci, S. Cetin, and N. I. Altintas, "How to Process [Business] Processes", Integrated Design and Process Technology, San Diego – USA, 2006.

Posterler

Always Altyapı Kullanan İş Uygulamalarında Mimari Değişim

Savaş Alparslan

Model Bilgi İşlem Hizmetleri Sanayi ve Ticaret Ltd. Sti,
Yıldızposta Caddesi Ayyıldız Sitesi No :28 - 58, Gayrettepe
34394 İstanbul, Türkiye
savas.alparslan@mbi.com.tr

Özet

1995'de geliştirilmiş olan Always altyapısı, 2 katmanlı istemci-sunucu mimarisinde iş uygulamaları geliştirilmesini sağlıyordu. Zamanla Internet ve B2B teknolojisindeki gelişmeler ile iş uygulamalarının kendi başına çalışan değil, web uygulamaları, web services gibi başka teknolojiler ile etkileşebilecek şekilde konumlandırılmasını zorunlu kıldı. Bunun için altyapı 3 katmanlı mimariye dönüştürüldü. Bu dönüşüm gerçekleştirilirken varolan iş uygulamalarının etkilenmemesine dikkat edildi.

Yazılım Endüstrisi'nde Yeniden Kullanım Kavramları

H. Burcu Özbek Terzi

ASELSAN *A.Ş.*
Kıdemli Uzman Mühendis
bozbek@aselsan.com.tr

C. Tolga Yurdakul

ASELSAN *A.Ş.*
Uzman Mühendis
ctolga@aselsan.com.tr

Özet

Yeniden kullanım Yazılım Mühendisliği disiplininin üzerinde uzunca süreden beri çalıştığı bir konudur. Bu makalede yeniden kullanım kavramlarının – mimari kalıpların, tasarım kalıplarının, yazılım bileşenlerinin, uygulama çerçevelerinin, yazılım ürün hattının – evrimi ve birbirleri ile olan ilişkileri incelendi. Kavramların endüstride kullanımı tartışılarak, olgunlaşmalarının yazılım ürün hattı açısından önemi vurgulandı.

Component Variants, Incremental Programming and Smoothing

Yusuf Altunel

İstanbul Kültür University,
Department of Computer Engineering
Bakirköy - İstanbul /TURKEY
y.altunel@iku.edu.tr

Mehmet R. Tolun

Çankaya Üniversitesi,
Department of Computer Engineering,
Balgat - Ankara / TURKEY
tolun@cankaya.edu.tr

Abstract

Our studies show us that inheritance reduces the maintainability and makes the incremental software development painful. To solve the problem we suggest a component-based methodology based on the composition and component variants to remove the inheritance from the design model conserving its benefits. Component variants are our new software units used to provide functional variation, support incremental development, and eliminate the bulky code from the software systems. The variants have the capability to replace the derived classes in object-oriented counterpart. In this paper we provide a grammar to algebraically define the components and variants and define a set of activities to identify and optimize the components, subcomponents and their variants.

Gömülü Yazılım Mimarileri'ne Bakış ve ASELSAN Yaklaşımı

Gülgün Kaderli

ASELSAN *A.Ş.*
Kıdemli Uzman Mühendis
guvenc@mst.aselsan.com.tr

Tolga İpek

ASELSAN *A.Ş.*
Kıdemli Uzman Mühendis
tipek@mst.aselsan.com.tr

Özet

ASELSAN, çok küçük cihazlardan, çok sayıda farklı sistemin bir araya gelmesiyle oluşan büyük boyutlu ve karmaşık sistemlere kadar geniş bir yelpazede urun geliştirmektedir. Günümüzde pek çok alanda karsımıza çıkan gömülü yazılımlar, ASELSAN'ın geniş urun yelpazesinde önemli bir yer teşkil etmektedir. Bu bildiride, gömülü yazılım mimarileri hakkında yapılan araştırmalardan edinilen bilgiler, projelerde edindiğimiz tecrübelerle sentezlenerek, ASELSAN' da Silah Sistemleri Yazılımlarının geliştirilmesinde kullanılmak üzere oluşturulan Gömülü Yazılımlar Referans Mimari Modeli'nin tanıtımı yapılacaktır.

Bilgi Sistemlerinin Birlikte Çalışabilirliği: Muharebe Yönetim Dili

Ahmet Kandakoğlu

Endüstri Mühendisliği Bolumu
İstanbul Teknik Üniversitesi, Maçka,
İstanbul 34367, Türkiye
kandakoglu@itu.edu.tr.

İlker Akgün
Endüstri Mühendisliği Bolumu
İstanbul Teknik Üniversitesi, Maçka,
İstanbul 34367, Türkiye
akguni@itu.edu.tr.

Özet

Son yıllarda baş döndürücü bir hızla gelişen teknoloji, iş süreçlerinde köklü değişmelere sebep olarak, ulusal ve uluslararası organizasyonların hem iç hem de dış iş süreçlerini sahip oldukları bilgi sistemleri aracılığıyla gerçekleştirmesini yaygın hale getirmektedir. Fakat farklı yazılım mimarileriyle birbirinden bağımsız olarak geliştirilen bu bilgi sistemlerinde veri modeli ve standartları uyuşumu göz önüne alınmadığından, birlikte çalışabilirlik güncel ve kritik bir sorun olarak karsımıza çıkmaktadır. Bu makalede bilgi sistemlerinin birlikte çalışabilirliği temel düzeyde incelendikten sonra Modelleme Simülasyon (M&S) ve Komuta Kontrol Bilgi Sistemlerinin (KKBS) birlikte çalışabilirliğini sağlamak üzere halen geliştirilmekte olan Muharebe Yönetim Dili (Battle Management Language – BML) tanıtılmaktadır. BML, askeri görevleri icra eden birlik ve teçhizatların komuta kontrolü ile durum farkındalığı ve paylaşılmış, ortak taktik resmi elde etmek için kullanılan bir dildir. BML bir bakıma gerçek ve simüle edilmiş birlikler ile gelecekteki robot sistemlerin kullanabileceği emir, istek ve raporların standart bir gösterimi olarak da düşünülmektedir. Bununla birlikte, KKBS, M&S ve robot sistemlerin ortak bir veri modeli doğrultusunda geliştirilmesi ve böylelikle birlikte çalışabilirlik yeteneğinin daha yazılım mimarisi tasarım aşamasında kazanılmasını sağlamaktadır. Ayrıca, bu yaklaşım farklı bir bakış açısıyla ele alındığında, sadece askeri uygulamalar için değil, birçok bilgi sistemini barındıran büyük organizasyonlar için güzel bir örnek teşkil etmektedir.

Silah Sistemleri'nde Kullanılan Gömülü Yazılım Mimarileri ve ASELSAN'da Gerçekleştirilen Projelerde Edinilen Tecrübeler

Gökhan Kahraman,

ASELSAN *A.Ş.*
Mikrodalga ve Sistem Teknolojileri Grubu
Kıdemli Uzman Mühendis
gkahraman@aselsan.com.tr

Evrim Kahraman,

ASELSAN *A.Ş.*
Mikrodalga ve Sistem Teknolojileri Grubu
Uzman Mühendis
altas@aselsan.com.tr

Serkan Ceylan

ASELSAN *A.Ş*
Mikrodalga ve Sistem Teknolojileri Grubu.
Uzman Mühendis
ceylan@aselsan.com.tr

Özet

Bu bildiride, ASELSAN A.S. Mikrodalga ve Sistem Teknolojileri (MST) Grubu Silah Sistemi projelerinin birbiri ile benzer fonksiyonel gereksinimlere sahip gömülü yazılımları kapsamında, yazılım geliştirici açısından göz önünde bulundurulan kalite faktörlerinden, bu faktörler ışığında seçilen katmanlı mimariden (layered architecture) ve alınan mimari tasarım kararlarından bahsedilecektir. Sunulan mimarinin ilgili projelerde uygulanmasıyla edinilen tecrübeler, hem elde edilen kazanımlar hem de zaman içerisinde karşılaşılan sorunlar ve yapılan değişiklikler ile irdelenecektir.

YAZARLAR DİZİNİ