2. ULUSAL YAZILIM MİMARİSİ KONFERANSI '08

BİLDİRİLER KİTABI

Editörler

Yasemin TOPALOĞLU, Ali DOĞRU, Halil ŞENGONCA

UYMK'08

11-12 Eylül 2008

Ege Üniversitesi
Bilgisayar Mühendisliği Bölümü
İzmir

2.Ulusal Yazılım Mimarisi Konferansı Bildiriler Kitabı TÜBİTAK desteği ile basılmıştır.

İÇİNDEKİLER

ÖNSÖZ

Yazılım mimarisi, büyük ölçekli ve karmaşık yazılım sistemlerinin geliştirilmesinde üretkenliği ve kaliteyi artırarak, yeniden kullanılabilirlik gibi niteliklerin elde edilebilmesi için büyük önem taşımaktadır. Bu bağlamda, Ulusal Yazılım Mimarisi Konferansı(UYMK), yazılım mimarisiyle ilgili araştırma ve deneyimlerin tartışıldığı bir ortam oluşturarak, yazılım mimarisi tasarımı bilincinin geliştirilmesine ve bu alandaki araştırmaların artırılmasına katkıda bulunmayı amaçlamaktadır.

Yazılım mühendisliği alanında çalışan akademisyenler ve endüstri temsilcilerinin oluşturduğu UYMK'08 Program Kurulu'nun değerlendirmeleri sonucunda, konferansa gönderilen 34 bildiri arasından 22 bildiri sözlü olarak sunulmak üzere kabul edilmiştir. Bu bildiriler, yazılım geliştirme sürecindeki deneyimleri, belirli bir alana yönelik mimari uygulamaları, yeni nesil uygulamalar için mimari yaklaşımları ve mimari modelleme konularını içermekte ve hem yazılım endüstrisinin hem de akademik çevrelerin yazılım mimarisi alanındaki çalışmalarını örneklemektedir.

Ayrıca konferansta davetli konuşmacılar Dr.Alessandro Garcia, "*Pursuing Software Architecture Stability in an Unstable Contemporary World:New Findings and Challenges*" başlıklı sunumunda, değişen gereksinimler ortamında yazılım mimarisinin kararlılığını sağlamak için geliştirilen güncel yazılım mühendisliği yaklaşımlarını tartışmış, Dr.Naci Akkök ise "*Uygulama Mimarileri ve Ara Yazılımlarda Teknolojinin Durumu*" başlıklı sunumunda, servis tabanlı ve olay tabanlı yaklaşımların, uygulama sunucusu tasarımına etkilerini değerlendirmiştir.

Ege Üniversitesi Bilgisayar Mühendisliği Bölümü'nün ev sahipliğinde düzenlenen 2.Ulusal Yazılım Mimarisi Konferansı'nın tüm ilgililere yararlı olmasını diler, UYMK'08 Program Kurulu üyelerine, bildiri yazarlarına, davetli konuşmacılara ve bildiri kitabının basımına sağladığı destek için TÜBİTAK'a teşekkürlerimizi sunarız.

Düzenleme Kurulu Başkanı
Halil ŞENGONCA

Program Kurulu Eş Başkanları
Yasemin TOPALOĞLU, Ali DOĞRU

Düzenleme Kurulu Başkanı

Halil Şengonca, *Ege Üniversitesi*

Program Kurulu Eş Başkanları

Yasemin Topaloğlu, *Ege Üniversitesi*
Ali Doğru, *Orta Doğu Teknik Üniversitesi*

Program Kurulu

Naci Akkök, *Oslo Üniversitesi*
Mehmet Akşit, *Twente Üniversitesi*
Selim Akyokuş, *Doğuş Üniversitesi*
Levent Alkışlar, *Aselsan*
Meriç Aykol, *TBD İstanbul*
Turgay Aytaç, *Logo Business Solutions*
Fevzi Belli, *Paderborn Üniversitesi*
Ayşe Başar Bener, *Boğaziçi Üniversitesi*
Semih Çetin, *Cybersoft*
Oğuz Dikenelli, *Ege Üniversitesi*
Banu Diri, *Yıldız Teknik Üniversitesi*
Ali Doğru, *Orta Doğu Teknik Üniversitesi*
Sami Duman, *Aselsan*
Nadia Erdoğan, *İstanbul Teknik Üniversitesi*
Rıza Cenk Erdur, *Ege Üniversitesi*
Yenal Göğebakan, *Cybersoft*
Haluk Gümüşkaya, *Fatih Üniversitesi*
Oya Kalıpsız, *Yıldız Teknik Üniversitesi*
Alpay Karagöz, *Bilgi Grubu*
Ümit Karakaş, *İstanbul Kültür Üniversitesi*
Sefer Kurnaz, *Hava Harp Okulu*
Can Özturan, *Boğaziçi Üniversitesi*
Murat Tanık, *Alabama Üniversitesi*
Bedir Tekinerdoğan, *Twente Üniversitesi*
Yasemin Topaloğlu, *Ege Üniversitesi*
Özgür Ulusoy, *Bilkent Üniversitesi*
Murat Osman Ünalır, *Ege Üniversitesi*
Meltem Turhan Yöndem, *Orta Doğu Teknik Üniversitesi*

Yerel Düzenleme Kurulu

Bekir Afşar, *Ege Üniversitesi*
Ahmet Egesoy, *Ege Üniversitesi*
Özgür Gümüş, *Ege Üniversitesi*
Önder Gürcan, *Ege Üniversitesi*

DAVETLİ KONUŞMACILAR

Pursuing Software Architecture Stability in an Unstable Contemporary World: New Findings and Challenges

Dr. Alessandro Garcia

Computing Department, Lancaster University, United Kingdom
garciaa@comp.lancs.ac.uk

Abstract. Software requirements and organisational rules are increasingly volatile. On the other hand, the presence of software changes affecting multiple modules in software architectures is generally considered to be an indication of poor software evolvability. These widely-scoped design properties decrease long-term stability of the underlying software architecture as many ripple effects and errors are likely to manifest in the presence of changes. This recognition has recently directed the research goals of software engineering – aggressively developing contemporary modularisation techniques, such as aspect-oriented design, to support stable representation of concerns that typically crosscut conventional architectural decompositions, such as object-oriented, MVC, and layered architectures. However, little is known about the actual contributions of aspect-oriented software architectures to improve design stability. In this context, this talk is going to discuss emerging empirical knowledge obtained about the positive and negative influences of aspectual decompositions on architecture stability. The talk will also reflect upon the influence exerted by aspect-oriented programming mechanisms in the architecture stability of concerns in evolving stand-alone and product-line architectures. The discussion will be driven by comparative analyses of aspectual and non-aspectual decompositions based on different architectural styles, such as layers, MVC, and blackboard architectures. Finally, this lecture will discuss some emerging challenges on the empirical assessment of aspect-oriented software architectures.

Uygulama Mimarileri ve Ara Yazılımlarda Teknolojinin Durumu

Dr. Naci Akkök

Oracle Nordics & Institute of Informatics, University of Oslo
nacia@ifi.uio.no

Özet. Uygulama mimarileri uzunca bir süredir JEE gibi bileşen odaklı yaklaşımlardan etkilenmektedir. Uygulamaya destek verme amaçlı ara yazılımlar da (örnegin application server'lar) doğal olarak bu tarz teknolojilerin sundugu/desteklediği soyutlamalar üzerine kurulmaktadır. Ama İş Süreci Yönetimi (BPM) destekli Servis Odaklı Mimariler (SOA) ve Olay Odaklı Mimariler (EDA) yeni nesil uygulama ara yazılımlarına ve uygulama referans mimarilerine temel olabilecek yazılım dillerinden ve teknolojilerinden bağımsız yeni soyutlamalar sunmaktadır. Bu konuşmada, uygulama mimarilerindeki ve ilintili ara yazılımlarındaki bu eğilimleri Oracle ve BEA Systems gibi söz sahibi yazılım firmalarının kısmi Ar&Ge sonuçlarından ve NESSI (Networked European Software & Services Initiative, http://www.nessi-europe.com/Nessi/) gibi kuruluşların çalışmalarından örnekleyerek özetlemeye çalışacağız.

YAZILIM GELİŞTİRME SÜRECİNDE MİMARİ

Büyük Ölçekli Bir Yazılım Sistemi için Mimari Dokümantasyon Süreci Deneyimleri

Oğuz İçoğlu[1], Oğuz Dikenelli[2], Onur Destanoğlu[3], Sevgi Akgün[4]

[1,2,3,4] TÜBİTAK, Marmara Araştırma Merkezi, Bilişim Teknolojileri Enstitüsü, 41470, Gebze, Kocaeli
[1] oguz.icoglu@bte.mam.gov.tr, [2]oguz.dikenelli@bte.mam.gov.tr,
[3] onur.destanoglu@bte.mam.gov.tr, [4]sevgi.akgun@bte.mam.gov.tr

Özet. Yazılım sistemlerinde, mimari tasarımın oluşturulma süreci, üzerinde dikkat edilmesi gereken önemli bir unsurdur. Özellikle büyük ölçekli sistemlerde mimarinin öneminin daha da arttığını görmekteyiz. Bunun en büyük nedeni, yazılım mimarisinin tüm projeyi paydaşları ile birlikte bir arada tutmaya yarayan bir yapıştırıcı görevi görmesi. Mimarisi olmayan bir projenin tökezleme ve başarısızlıkla sonlanma riskleri yükselmektedir. Diğer taraftan, mükemmel bir mimari bile eğer iyi anlatılamamışsa ve anlaşılamamışsa, yani diğer bir deyişle iyi dokümante edilmemişse, başarılı bir mimari tasarım sürecinin varlığından bahsedilemez.

Günümüzde, yazılım mimarilerinin yapılandırılmasına yönelik değişik yöntemler bulunmaktadır. Bu yöntemlerden uygun olan bir tanesi, TÜBİTAK, Marmara Araştırma Merkezi, Bilişim Teknolojileri Enstitüsü'nde gerçekleştirilen büyük ölçekli bir benzetim projesinde kullanılmaktadır. Projenin büyük ölçekli olması ve gevşek-bağlı bir yapıda dağıtık olması bir mimari dokümanın varlığını zorunlu kılmıştır. Dokümantasyon için uygun yöntem, gereksinimler doğrultusunda belirlenmiş ve projeye adaptasyonu amacıyla modifiye edilmiştir. Bu makalede uygun yöntemin seçilmesi ve modifiye edilmesi sırasında alınan kararlar ve projeye uyarlanması sırasında kazanılan deneyimler anlatılmıştır. Buna ek olarak, mimari dokümanının projedeki paydaşlar tarafından ne şekilde kullanıldığı açıklanmış ve doküman üzerinden gerçekleştirilmesi planlanan ileriye dönük çalışmalar belirtilmiştir.

1 Giriş

Yazılım mimarisi, sistem ve onu geliştiren proje için çoğu zaman bir ana plan görevi görmektedir. Mimari, sistemin yapısını ve alt yapılarını belirtir. Yapıların meydana geldiği öğeleri ve öğeler arasındaki ilişkileri net olarak tanımlar. Bunun dışında, geliştirilecek sistemin performans, güvenlik, modifiye-edilebilirlik gibi bazı kalite niteliklerini sağlaması gerekebilir. Yazılım mimarisi, sistemin bu gibi kalite özelliklerini belirler. Kaliteli bir sistem ancak mimaride belirlenmiş bir vizyonla gerçeklenebilir.

Ancak en mükemmel mimari bile anlaşılır bir şekilde ifade edilmediği sürece bir işe yaramamaktadır. Bundan dolayı mimariyi açık, "yeterince" detaylı ve organize bir şekilde dokümante etmek önemlidir. Bu doğrultuda, TÜBİTAK, Marmara Araştırma Merkezi, Bilişim Teknolojileri Enstitüsü'nde gerçekleştirilen büyük ölçekli bir benzetim projesinde mimari dokümantasyon süreci başlatılmıştır. Sürecin amacı, sistemin mimari

dokümanını oluşturmak ve bu dokümanın, sistemin gerçeklenmesine ve devamlılığının sağlanmasına hizmet etmesini sağlamaktır.

2 Motivasyon

Proje çerçevesinde gerçekleştirilecek sistem büyük ölçekli bir benzetim sistemidir. Sistemin ana amacı, belirli tipteki gemilerin modellenerek benzetiminin yapılmasıdır. Bu şekilde, gemi personelinin eğitiminin çok daha verimli ve ekonomik hale gelmesi mümkün olmaktadır. Ancak bir gemide radardan, motorlara kadar bir çok alt bileşen bulunmaktadır. Bu da, sistemin büyük ölçekli olması ve gevşek-bağlı bir yapıda tasarlanmasını zorunlu kılmıştır. Böyle bir sistemin gerçeklenmesinde ve devamlılığının sağlanmasında üç temel sorunla karşılaşılmıştır.

Öncelikle, geminin alt bileşenleri ayrı birer modül olarak tasarlanmıştır, ve her modülün diğerlerinden farklı bir yapısı ve gereksinimleri bulunmaktadır. Bundan dolayı, proje geliştirme ekibine yeni katılan üyelere sistemi tanıtmak zahmetli ve zaman alan bir süreç haline gelmiştir.

Ayrıca, sistemin geliştirilme süreci çeşitli fazlar üzerinden planlanmıştır. Her bir fazda, sistemin belirli bir işlevselliğe ulaşmış bir sürümü elde edilmektedir. Bir sonraki faza geçişte, sistem üzerinde yapılacak eklemeleri ve değişiklikleri planlamak, bunların mevcut yapıya etkisini görebilmek giderek zor bir hale gelmiştir.

Son olarak, daha önce de belirtildiği gibi, sistemden ve onu oluşturan modüllerden belirli kalite niteliklerini sağlaması beklenmektedir. Kimi modüllerin birim zamanda belirli bir miktar veriyi işlemek gibi performans hedefleri, kimi modüllerin de sahip oldukları verilere erişimi kontrol etmek gibi güvenlik hedefleri olabilmektedir. Kimi modüllerin ise yapısı gereği kolay modifiye-edilebilir olması gerekmektedir. Bu gibi kalite niteliklerinin analizini yapmak ve sistemin tıkandığı yerleri bulmak, ya da ileride çıkabilecek sıkıntıları öngörebilmek, sistemi geliştirenler için önem taşımaktadır.

Yukarıda belirtilenler ışığında sırasıyla, eğitim, planlama ve analiz çalışmaları için baz oluşturacak bir altyapıya ihtiyaç duyulmaktadır. Bu görevi yerine getirecek en uygun platformun sistemin mimari dokümanı olduğuna karar verilmiş, ve böylece sistemin mimarisini dokümante etme ihtiyacı doğmuştur. Mimari doküman öncelikle bir eğitim aracı olarak hizmet etmektedir. İnsanlara sistemi tanıtmaya yardımcı olur. Bu insanlar yazılım ekibinin yeni üyeleri olabildiği gibi, dışardan katılan analistler veya yeni bir mimar olabilir. Aynı zamanda, paydaşlar arasında iletişimi sağlar. Mimarinin paydaşı, mimari üzerinde sermayesi ve çıkarı olan kişidir. En önemli paydaş genellikle projenin gelecekteki mimarıdır. Yeni mimar, seleflerinin projenin gelişimi esnasında hangi konularda takıldığını, hangi kararları, neden verdiklerini öğrenmek isteyecektir. Hedeflenen planlama çalışmalarının temelinde de, mimarinin diğer paydaşlarından olan proje yöneticileri, tasarımcılar ve geliştiriciler arasındaki iletişimin kurulması yatmaktadır. Son olarak, mimari doküman sistem analizi için bir temel oluşturur. Sistemin kalite hedefleri olabilir. Tasarımın bu hedefleri ne kadar karşılayabildiği ile ilgilenenler için mimari doküman bir rehber özelliği taşımaktadır.

2.1 Benzetim Sisteminin Yapısı

Teknolojideki hızlı ilerlemeler ve güçlü bilgisayarların geliştirilmesi benzetim sistemlerinden beklenenleri arttırmıştır. Bununla birlikte aynı ortamda koşan benzetimlerin birlikte çalışılabilirliğinin sağlanması da son yıllarda üzerinde yoğun çalışmalar yapılan alanlardan biri olmuştur. Amerikan Savunma Bakanlığı tarafından desteklenen çalışmalar sonucunda, bir dağıtık ve interaktif benzetim sistemi geliştirme yöntemi oluşturulmuştur. HLA (High Level Architecture) adı verilen bu yapı daha sonra IEEE standardı haline gelmiş ve IEEE Standard 1516 [1] ismi altında düzenlenmiştir. Bu yapı temelde dağıtık benzetimlerin bir arada çalışabilmesi için sunulmuş bir mimaridir.

HLA, dağıtık benzetim sistemlerinin geliştirilmesi için esnek bir altyapı sunmaktadır. Bu sayede sistemin farklı modülleri bir arada çalışabilmekte ve proje içinde tekrar-kullanılabilmektedir. Diğer bir deyişle, gerçeklenen benzetim sistemi, modellenen varlıklarda yapılan değişimlerden en az şekilde etkilenmekte ve kolayca değiştirilerek yeniden kullanılabilmektedir. Ayrıca modüller bilgisayar sisteminden bağımsız olarak birbirlerine bağlanabilmektedir.

Bu yapı içerisinde yer alan benzetim modüllerine federe adı verilir. Federler kendi aralarında özel bir haberleşme altyapısı üzerinden etkileşimde bulunurlar. Bu altyapıya RTI (Runtime Infrastructure) adı verilmektedir. Federeler arasında mesajların iletiminden bu çalışma-anı altyapısı sorumludur. Gerçekte bir arakatman görevi gören RTI, federelerin bilgisayar sisteminden bağımsız bir şekilde bağlanmasını sağlamaktadır. Tüm federeler, bir arada çalışarak federasyonu, yani benzetim sistemini oluştururlar [2].

Proje çerçevesinde gerçeklenen benzetim sisteminde, geminin modellenmesi gereken çeşitli alt bileşenleri birer federe olacak şekilde tasarlanmıştır. Bu yapı içerisinde 24 adet federe oluşturulmuştur. Her federe farklı karmaşık modeller içermekte ve farklı gereksinimlere sahip bulunmaktadır. Buna bağlı olarak her federenin de farklı bir mimarisi vardır. Federeler kendi ihtiyaçlarına özgü değişik üçüncü parti sistemlerden yararlanmaktadır. Tüm bu etkenler eğitim ve planlama süreçlerini daha karmaşık bir hale getirmektedir.

2.2 Geliştirme Süreci

Daha önce de belirtildiği gibi, sistemin geliştirilme süreci çeşitli fazlar üzerinden planlanmıştır. Her bir fazda, sistemin belirli bir işlevselliğe ulaşmış bir sürümü elde edilmektedir. Ayrıca, fazlar içerisinde de döngüsel (iteratif) bir geliştirme süreci izlenmektedir. Faz içinde, gerçeklenmesi planlanan işlerin yoğunluğuna göre, üç veya dört döngü tamamlanmaktadır. Her döngüde federeler test edilmekte, federelerin tümleştirilmesi gerçeklenmekte ve sonrasında federasyonun testi yapılmaktadır.

Bu şekilde, aşamalı büyüme adı verilen bir geliştirme paradigması yürütülmektedir. Projenin daha ilk safhalarında, sistemin alt kümesi için ön-ürün geliştirme, tümleştirme ve test yapılabilmektedir. Sistemin başta fazla bir işlevselliği yoktur ama yeni modüller eklendikçe aşamalı olarak işlevsellik gelişir. Her aşamada (döngüde), alt küme, yeni eklenen modüllerle beraber tümleştirme ve test süreçlerini tekrar yaşar.

Ancak, her döngüde sisteme yeni işlevler ve kalite gereksinimleri eklenmektedir. Bu da yazılım geliştirme sürecini zorlayan önemli bir etkendir. Bundan dolayı, sistemin mimarisini dokümante etme ihtiyacı her döngüde artmaktadır.

Yukarıdaki yazılım geliştirme sürecine ek olarak, HLA tabanlı benzetim sistemlerinde federeler arası istenen veri akışını kurabilmek için IEEE tarafından standart bir federasyon geliştirme süreci tanımlanmıştır. FEDEP (Federation Development and Execution Process) [3] adı verilen bu sürecin ilk adımı, gerçek dünyada işlerin nasıl yürüdüğünü gösteren bir kavramsal model oluşturmaktır. Bu model, tasarımdan teste kadarki tüm geliştirme sürecinde bir rehber görevi görmektedir. Bu modelden yararlanılarak BOM (Base Object Model) adı verilen kavramsal senaryolar çıkarılır. Bu senaryolar ise federelerin ve federasyonun tasarım ve test aşamasında kullanılmaktadır [2]. Geliştirme sürecinde yeni döngülere geçildikçe, federasyon senaryolarındaki etkileşim ve senkronizasyon gereksinimleri artmakta, bu da mimari dokümantasyon ihtiyacını zorunlu kılmaktadır.

3 Mimari Dokümantasyon Süreci

Sistem mimarisinin oluşturulmasında, yukarıda belirtilen FEDEP sürecinde tanımlanan kavramsal model temel alınmıştır. Kavramsal model içerisinde ortaya çıkan ve sistemin yürütmesi gereken görevler, mimari tasarıma kaynak olmaktadır. Ayrıca, daha önce belirtildiği gibi, her federe farklı karmaşık modeller içermekte ve farklı gereksinimlere sahip bulunmaktadır. Buna bağlı olarak her federenin farklı bir mimarisi olması gerekmektedir. Bundan dolayı, dokümantasyonda federeler bazında bir ayrıştırma olacağı, dokümantasyon yönteminin seçilmesi sürecinden önce belirginleşmiştir.

3.1 Yöntemin Belirlenmesi

Yazılım mimarilerinin dokümantasyonuna yönelik çeşitli yöntemler vardır. Ancak tüm yöntemler genelde görünüm (view) adı verilen bir yapı üzerine kuruludur. Yazılım mimarisi karmaşık bir varlıktır ve tek boyutlu bir biçimde ifade edilemez. Farklı görünümler biraraya gelerek mimariyi oluştururlar. Sistemde, birbiriyle alakalı belirli öğeler ve ilişkiler bir küme altında toplanabilirler. Görünümler, bu farklı bakış-açılarıyla oluşturulmuş kümelerin bir temsilidir. Mimariyi dokümante etmek aslında uygun görünümleri belgelemektir. İşte bu uygun görünümler değişik yöntemlerde değişik şekilde tanımlanmaktadır. Bunlar içinden örnek vermek gerekirse; IBM'in Rational Unified Process yöntemi [4], Siemens'in Dört-Görünüm yöntemi [5], Amerikan Savunma Bakanlığı'nın C4ISR (Command, Control, Communication, Intelligence, Surveillance, Reconnaissance) yöntemi [6] ve IEEE'nin 1471-2000 mimari dokümantasyon yöntemi [7] sayılabilir. Bunlara ek olarak, daha önce yazılım süreçlerinde olgunluğun tanımlanması için CMMI (Capability Maturity Model Integration) sertifikasyonunu yaratmış Carnegie Mellon Üniversitesi, Yazılım Mühendisliği Enstitüsü'nün Görünümler ve Ötesi yöntemi bulunmaktadır.

Mimari dokümantasyon sürecinin ilk adımında, mevcut yöntemleri incelemek ve içinden uygun olanını seçmek amacıyla bir üst mimari grup oluşturulmuştur. Bu grubun yaptığı çalışma sonucunda gerek Bilişim Teknolojileri Enstitüsü'nde yürütülen CMMI çalışmaları ile uyumu, gerekse her türlü projeye uyarlanabilen jenerik yapısı ile

Carnegie Mellon Üniversitesi, Yazılım Mühendisliği Enstitüsü'nün Görünümler ve Ötesi yöntemi [8] seçilmiştir.

Görünümler ve Ötesi. Bu yöntemde, görünümlerin sayısı konusunda bir sınırlama olmasa da, görünümler geniş çizgilerle sınırlandırılmıştır. Bu yönteme göre mimarlar, yazılımları için üç konu üzerinde eşzamanlı olarak düşünmek zorundadır:

1. Yazılım, gerçekleme birimleri açısından nasıl yapılandırılmış?
2. Yazılım, çalışma-anı davranışları açısından nasıl yapılandırılmış?
3. Yazılım, çalışma ortamındaki yazılımdışı yapılarla nasıl bir ilişki içinde?

Yöntemde, her görünümün, yukardaki üç konudan birine denk düşmesinin uygun olacağı öngörülmüştür. Böylece, görünümler üç görünümtipi altında sınıflandırılmıştır:

1. Modül görünümtipi (module viewtype)
2. Bileşen-ve-bağlantı görünümtipi (component and connector viewtype)
3. Dağıtım görünümtipi (allocation viewtype)

Modül Görünümtipi. Modül görünümleri sistemin tasarım-anına ilişkin bilgi verir. Bu görünümtipinin öğeleri modüllerdir. Yazılımda modül kavramı ilk olarak 1970'lerde ortaya çıkmıştır. Modül, çeşitli hizmetler sunan yazılım birimleri olarak ifade edilir. İlk olarak prosedürler ve fonksiyonlar modülleri oluşturmaktaydı. Giderek yayılan nesne-yönelimli tasarım paradigması ile sınıflar ve katmanlar modülü oluşturmaya başladı. Yazılımın modüllere ayrılması ile büyük sistemler daha kolay yönetilebilir hale geldi. Bir mimari dokümanının hiçbir *modül görünümü* olmaması çok az rastlanan bir durumdur. Böyle bir durum, tasarım ile ilgili hiçbir bilgi sunulmadığı anlamına gelmektedir. Genelde her dokümanın en az bir modül görünümtipinden bir görünümü bulunur.

Bileşen-ve-Bağlantı Görünümtipi. Bileşen-ve-bağlantı görünümleri çalışma–anına ilişkin öğelerden oluşur. Bu öğeler genellikle; paralel süreçler, istemciler, sunucular veya veri-depolarıdır (bileşenler). Ayrıca, iletişim bağlantıları, veri akışı veya ortak veri alanına erişim gibi etkileşimler de (bağlantılar) bu görünümtipinin öğeleridir. Bir *bileşen-ve-bağlantı görünümü*, çalışma-anı varlıklarının ve onların olası etkileşimlerinin bir fotoğrafıdır. Çalışma-anına ilişkin olduğundan (tasarımdan farklı olarak) bazı bileşenler görünüm içinde birden çok yerde görülebilir.

Dağıtım Görünümtipi. Daha önceki iki görünümtipi de yazılım mimarisi üzerine kuruluydu. Dağıtım görünümtipi; donanım, dosya sistemleri, çalışma ekibi gibi yazılımdışı unsurların (ortam öğelerinin) yazılım mimarisi (yazılım öğeleri) ile etkileşimini gösterir.

Sistemin mimari dokümantasyon sürecinde geriye kalan çalışma, her federe için yukardaki görünümtiplerinden uygun görünümleri seçip, onları belgelemektir. Sonra bu belgelere, tüm görünümleri birleştirecek ve federasyonu da kapsayacak ek bir belge daha eklenir ve mimari doküman ortaya çıkar. Ancak yukarıda bahsedilen yöntemlerden hiçbiri, sistemi tasarlayan mimarı bir tasarım aracına veya modelleme diline yönlendirmez. Yöntemler araçtan ve dilden bağımsız bir şekilde oluşturulmuştur. Mimari dokümanda ortak bir organizasyon ve ortak bir dil oluşturmak, dokümanın tutarlılığı ve açıklayıcılığı açısından önemlidir. Bundan dolayı, dokümantasyon sürecinin ikinci adımında ortak bir şablon oluşturulmuştur.

3.2 Şablon Oluşturulması

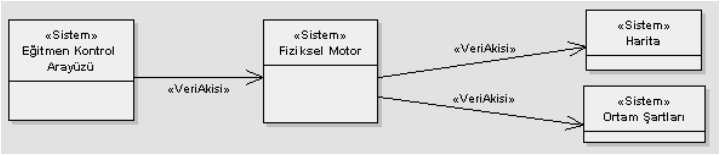
Yukarıda da bahsedildiği gibi, şablon, dokümanın tutarlılığı ve açıklayıcılığı açısından önemlidir. Buna ek olarak, mimarlara, mümkün olduğunca pratik ve hızlı bir şekilde dokümanı hazırlamak için bir altyapı görevi sunmalıdır. Bu amaçla, şablonun hazırlanması için bir alt mimari grup oluşturulmuştur. Bu alt grup, sistemde yer alan üç federenin yazılım ekibinden seçilmiştir. Federeler seçilirken, tasarımlarında farklı modeller olmasına ve farklı kalite gereksinimlerini yerine getirmesine dikkat edilmiştir. Bu şekilde, yöntem projeye uyarlanırken, mümkün olduğunca çok olasılıkla önceden karşılaşılmış olması hedeflenmektedir.

Öncelikle bu alt gruba seçilen yöntemi anlatan bir eğitim verilmiştir. Daha sonra, yöntem doğrultusunda her federenin uygun görünümleri belirlenmiş ve şablon oluşturulmaya başlanmıştır. Dokümantasyon aracı olarak Enterprise Architect programının 6.5'inci sürümü kullanılmıştır. Modelleme dili olarak, artık tüm projelerde ortak bir dil haline gelen UML (Unified Modelling Language)'den yararlanılmıştır. Daha önceden de belirtildiği gibi, yöntem belirli bir notasyon sunmamaktadır. Bundan dolayı, her görünümtipi için bir UML profili oluşturulmuştur. Bu profiller kullanılarak da Enterprise Architect'de bir şablon oluşturulmuştur.

3.3 Örnek bir Federe Mimarisinin Oluşturulması

Mimari dokümantasyon sürecinin bir sonraki aşaması, oluşturulan şablonun tüm yazılım ekibine yaygınlaştırılmasıdır. Ancak herkesin yöntemi ve şablonun kullanımını daha rahat anlayabilmesi açısından bir federenin örnek uygulamasının yapılmasına karar verilmiştir. Seçilen örnek federe, benzetim sisteminde geminin motor alt bileşenine denk gelmektedir. Bu alt bileşenin federasyon içindeki federe adı "fiziksel motor"dur. Örnek olması açısından basitleştirilmiş hali aşağıda anlatılmaktadır.

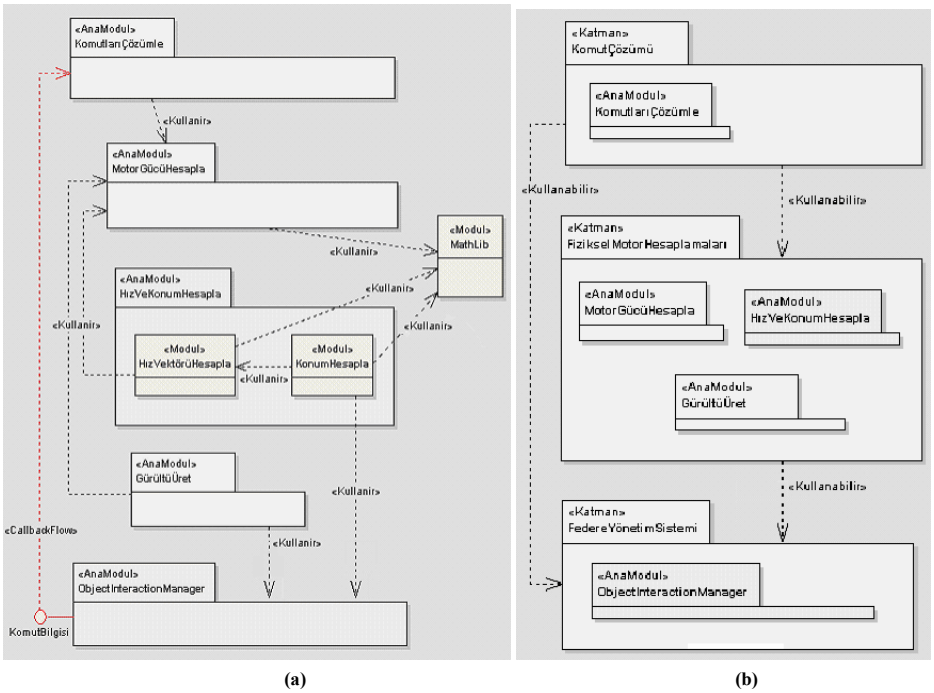
Bağlam. Dokümanda öncelikle mimarisi oluşturulan sistemin bağlamı verilir. Görünümlerin ilgi alanı, sistem içindeki modül ya da bileşen gibi mimari öğeler ile sınırlıdır. Federe hakkında genel bilgi vermez. Federenin çevresi ile olan etkileşimi hakkında bilgiler bu bölümde belirtilir. Sistemin çözmekle sorumlu olduğu ana problem açıklanır. Şekil 1'de Fiziksel Motor federesinin diğer federelerle olan ilişkisi gösterilmektedir. Burada belirtilen ilişki türü "veri akışı"dır. Eğitmen Kontrol Arayüzü'nden gelen seyrüsefer komutları alınır. Federe içerisinde motorun üreteceği güce göre geminin hızı ve konumu hesaplanır. Konum bilgisi gemiyi harita üzerinde gösterebilmesi için Harita federesine gönderilir. Aynı anda, motorun üreteceği gürültü hesaplanır ve Ortam Şartları federesine iletilir. Açıklamalar "destekleyici doküman" adı verilen bilgi notları olarak Enterprise Architect'deki ilgili alana eklenmektedir.



Şekil 1. Fiziksel Motor federesinin bağlamı.

Modül Görünümleri. Modül görünümtipleri için yöntem dört farklı görünüm tanımlamıştır. Her görünümde öğeler modüllerdir ancak aralarındaki ilişki değişmektedir. Bunlardan ilki ayrıştırma görünümüdür. Burada tanımlanan ilişki "parçasıdır" ilişkisidir. Sistemin modüllere ayrışmasını gösterir. Sistemin tasarımının ana resmini sunmak için ideal bir belgedir. Örnek federe için ayrıştırma görünümü sunulmamıştır. Ayrıştırmalar, yöntemde tanımlanan bir sonraki görünüm olan kullanım görünümü içinde verilmiştir (Şekil 2 (a)). Kullanım görünümü, modüller arasındaki "kullanım" ilişkisini, yani modüllerin bağımlılığını gösterir. Geliştiricilere, kendi kısımlarının doğru çalışabilmesi için hangi modüllerin varolması gerektiğini belirtir.

Yöntemin tanımladığı üçüncü görünüm katmanlı görünümdür. Burada istisnai olarak öğeler modüller değil, katmanlardır. Aradaki ilişki ise "kullanabilir" ilişkisidir. Katmanlı görünüm ile ifade edilmiş sistemlerin modifiye-edilebilirlik ve taşınabilirlik gibi önemli özellikleri vardır. Diğer katmanlar için "kullanabilir" izni tanımlı olmayan kısımlar rahatlıkla, sistemin geri kalanının yapısını bozmadan değiştirilebilir. Fiziksel Motor federesinde üç katman tanımlanmıştır. Bunun amacı, RTI haberleşme altyapısındaki ya da modellenen gemi tipindeki değişimin federeye etkisinin azaltılmasıdır (Şekil 2 (b)). Yöntem son olarak, sistemdeki türetmeleri göstermek amacıyla bir genelleştirme görünümü tanımlamıştır. Bu görünümle ifade edilen sistemlerin tekrar-kullanılabilirlik özelliği bulunmaktır. Örnek federede böyle bir yapı mevcut değildir.



Şekil 2. Fiziksel Motor federesinin modül görünümleri: (a) Kullanım, (b) Katmanlı Görünüm.

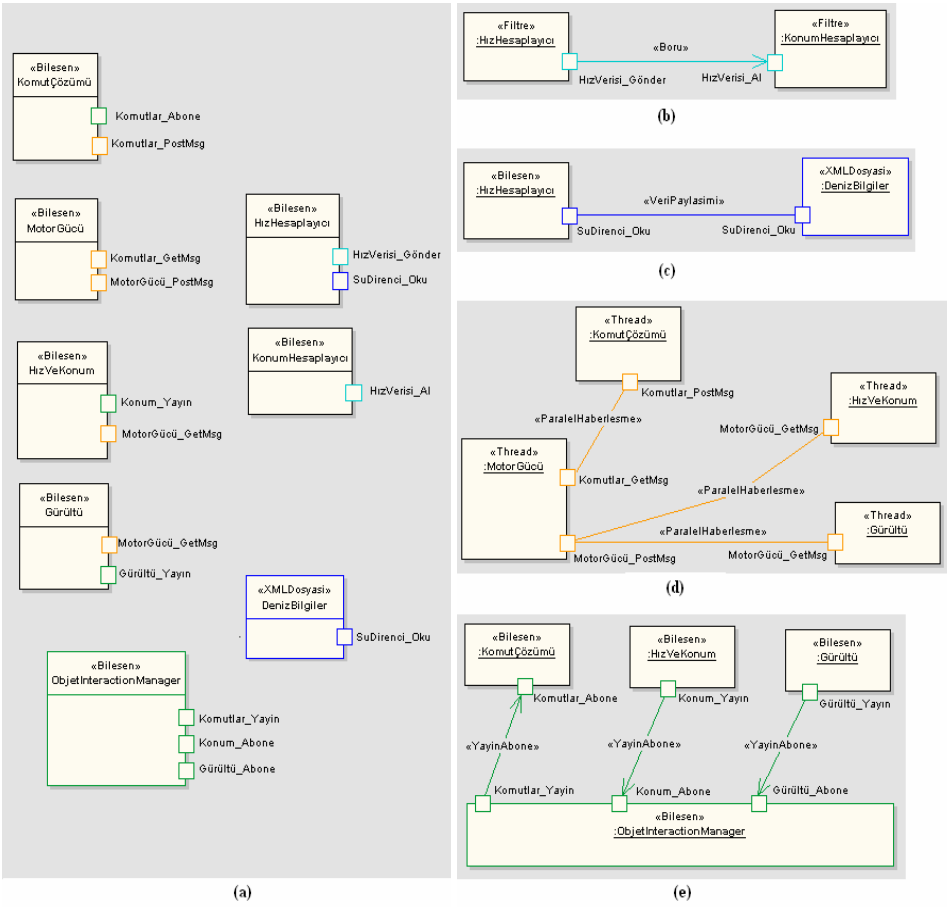
Bileşen-ve-Bağlantı Görünümleri. Çalışma-anı'na ilişkin öğelerden oluşur. Yöntem altı görünüm tanımlamıştır. Her görünüm farklı öğelerden oluşmaktadır. Ancak ilişki aynıdır. "Ekleme" ilişkisi ile bileşenler bağlantılar üzerinden birbirlerine eklenirler. Şekil 3 (a)'da örnek federe için bütün bileşenler tanımlanmıştır. Daha sonra ilgili görünümlere buradan taşınmıştır. İlk görünüm (Şekil 3 (b)) boru-ve-filtre görünümüdür. Bileşen tipi filtre, bağlantı tipi boru'dur. Bir filtre, bir ya da daha fazla borudan gelen veriyi dönüştürür ve sonuçları diğer bileşenlere aktarır.

İkinci görünüm paylaşımlı-veri'dir (Şekil 3 (c)). Bileşenler arası gerçekleşen etkileşim, kalıcı verinin (sistemin kapatılmasıyla kaybolmayan verinin) alıp-verilmesidir. Bileşenler, veriye erişen birimler (erişimciler) ve verinin saklandığı paylaşımlı-veri-depoları'dır. Üçüncü görünüm ise haberleşen süreçler'dir (Şekil 3 (d)). Paralel çalışan bileşenlerin çeşitli bağlantı mekanizmaları aracılığıyla etkileşimde bulundukları görünümdür. Herhangi bir sistemdeki parallellikleri görmek için en ideal belgedir.

Dördüncü görünüm yayın/abone'dir (Şekil 3 (e)). Bu görünüm federelerin RTI arakatmanı üzerinden etkileşimini belirtir. Beşinci ve altıncı görünümler sırasıyla istemci/sunucu ve uçbirimden-uçbirime görünümleridir. İki görünüm de bileşenler arası hizmet alımı ve sunumu türünde etkileşimleri gösterir. Aralarındaki fark, birinin asimetrik, diğerinin simetrik oluşudur. Örnek federede böyle bir yapı olmadığından bu görünümler verilmemiştir.

Yukarıdaki altı görünüm de, sistemin çalışma-anına ait statik durumunu temsil eder. Dinamizmi göstermek için yöntemde ek olarak sekans görünümü tanımlanmıştır. Bu görünümde, bileşenlerin zaman içindeki çalışma ve etkileşim düzeni gösterilir.

Dağıtım Görünümleri. Dağıtım görünümleri için yöntem iki görünüm tanımlamaktadır. İlki, modüller ile organizasyonel birimlerin eşleştirildiği görev tahsisi görünümüdür. Diğeri de bileşenler ile donanım birimlerinin eşleştirildiği konuşlandırma görünümüdür. Örnek federenin basitleştirilmiş halinde bu görünümler verilmemiştir.



Şekil 3. Fiziksel Motor federesinin bileşen-ve-bağlantı görünümleri: (a) Bütün bileşenler ve bağlantı kapıları, (b) Boru-ve-Filtre, (c) Paylaşımlı Veri, (d) Haberleşen Süreçler, (e) Yayın/Abone Görünümü.

Görünümlerin Eşleştirilmesi. Görünümülerin birbirinden kesin çizgilerle ayrıldığını düşünmek gerçekçi olmaz. Bazen, iki görünümün kullandığı ortak bir öğe olabilir. Bazen de, bir görünümde yer alan bir öğe, kendi içinde ayrı bir görünüm şeklinde sunulabilecek alt öğelerden oluşabilir. Genelde tasarım-anındaki modüller, çalışma-anına denk düşen bileşenleri ile eşleştirilir. Görünümler arasındaki ilişki bir tablo üzerinden gösterilir. Şekil 4'de Fiziksel Motor federesi için görünümler eşleştirilmiştir.

	Modül Kullanım	B&B Haberleşen Süreçler
B&B Boru ve Filtre		1.2
B&B Haberleşen Süreçler	1.1	

1.1 Modül - Kullanım Görünümü ile B&B - Haberleşen Süreçler Görünümü Arasında Eşleme

Kullanım Görünümü Ögeleri	Haberleşen Süreçler Görünümü Ögeleri	İlişki
KomutlarıÇözümle	KomutÇözümü	
MotorGücüHesapla	MotorGücü	
HızVeKonumHesapla	HızveKonum	
GürültüÜret	Gürültü	

1.2 B&B- Haberleşen Süreçler Görünüm ile B&B - Boru ve Filtre Görünümü Arasında Eşleme

Haberleşen Süreçler Görünümü Ögeleri	Boru ve Filtre Görünümü Ögeleri	İlişki
HızVeKonum	HızHesaplayıcı	
	KonumHesaplayıcı	

Şekil 4. Görünümlerin eşleştirilmesi.

3.4 Yöntemin Projeye Uyarlanması

Mimari dokümantasyon sürecinin son adımı, yöntemin tüm projeye yaygınlaştırılmasıdır. Bu amaçla, tüm federelerin katılımıyla genel bir ekip kurulmuştur. Bu ekibe, mimari dokümantasyon yöntemi, yukarıda verilen örnek üzerinden detaylı olarak anlatılmıştır. Bu eğitimin sonucunda, her federe yukarıdaki şekilde kendi mimarisini kurmuştur. Bütün federe mimarileri Enterprise Architect programında tek bir dosya altında birleştirilmiştir. Ayrıca, bu dosyaya, bütün federeleri bir arada gösteren birleştirici bir belge eklenmiştir. Eklenen belge, aslında federasyonun, federeler bazında oluşturulan modül ve bileşen-ve-bağlantı görünümlerinden meydana gelmektedir. Modül görünümleri içerisindeki kullanım görünümü ile federeler arasındaki bağımlılıklar verilmiştir. Bileşen-ve-bağlantı görünümleri içerisindeki sekans görünümü ile de federelerin zaman içerisindeki çalışma düzeni gösterilmiştir. Her federe RTI üzerinde yayınladığı veya abone olduğu mesajları kendi görünümünde belirtmişti. Ancak bu mesajların dış dünyada kimden geldiğini veya kim tarafından kullanıldığını görmek mümkün değildi. Ek belgede, bütün federelerin yayınladığı mesajlar tek bir arayüz tanımlama görünümü üzerinden gösterilmiştir. Bu görünümdeki mesajlar, ilgili federenin yayın-abone görünümünden referans alınarak oluşturulmuştur. Daha önce bahsedilmemesine rağmen, yöntem içinde arayüzlerin tanımlanmasına ilişkin bir bölüm bulunmaktadır. Temelde bir tablo yapısında olan bu görünüm, arayüz üzerinden sağlanan hizmetleri (yayınlanan mesajları) ve alınan hizmetleri (abone olunan mesajları) açıklamaktadır.

4 Sonuçlar ve Gelecekteki Çalışmalar

Makalede, büyük ölçekli bir benzetim sisteminin mimari dokümantasyonu sürecinde geçirilen aşamalar anlatılmıştır. Öncelikle, projenin aşamalı büyüme üzerine kurulu geliştirme süreci tanıtılmış, bu doğrultuda mimari dokümantasyona olan gereksinim belirtilmiştir. Temelde gereksinimler üç grupta toplanmıştır: eğitim, planlama ve analiz. Geliştirme sürecinde, her aşamada genişleyen sistemin eğitimini vermek, gelişimini

planlamak ve kalite nitelikleri bakımından analizini yapmak giderek zor bir hale gelmektedir. Bu ihtiyaçlara cevap verecek en uygun platformun mimari doküman olduğuna karar verilmiş, ve bu doğrultuda, üç temel adımdan oluşan bir mimari dokümantasyon süreci planlanmıştır. Birinci adımda, bir ön ekiple, mevcut yöntemler içinden en uygunu seçilmiştir. İkinci adımda, genişletilmiş ön ekiple, seçilen yöntem projeye uygunlaştırılmıştır. Bunun sonucunda, ortak bir organizasyonun ve dilin kullanıldığı bir şablon oluşturulmuş ve örnek bir uygulama yapılmıştır. Son adımda, örnek üzerinden süreç tüm projeye yaygınlaştırılmıştır. Bu doğrultuda hazırlanan mimari doküman, mevcut durumda eğitim amacıyla kullanılabilmekte, ve geliştirme sürecinin ilerleyen fazlarında sistemin gelişimi planlanabilmektedir. Ancak mimari doküman, kalite niteliklerinin analizi açısından bir altyapı oluştursa da, tek başına bir analiz yöntemi sunmamaktadır. Çeşitli mimari dokümantasyon yöntemleri olduğu gibi, mimari analiz yöntemleri de bulunmaktadır. Bunlar içinden, şu anda üzerinde çalışılan ATAM (Architecture Tradeoff Analysis Method) yöntemi [9] ilgi çekici gözükmektedir. Yöntem, temelde, mimari dokümanda belirtilen kalite niteliklerini baz alarak ödünleşimleri (tradeoff) ve riskleri belirlemektedir. Bu şekilde, belli kalite niteliklerinin sağlanmasının sistemin diğer kalite özelliklerini nasıl etkilediği ve sisteme ne gibi riskler getirdiği çözümlenebilmektedir. Projenin ilerleyen aşamasında, bu çeşit bir yöntem ile sistemin analizinin yapılması planlanmaktadır.

Kaynakça

1. IEEE Std. 1516™-2000, IEEE Standard for Modeling and Simulation High Level Architecture - Framework and Rules.
2. Timar, Y., Taşdelen, İ., Akgün, S., Dikenelli, O. ve Bıkmaz, İ., "Using BOM in Conceptual Modelling of Distributed Simulation Projects", Simulation Interoperability Workshop (SIW), Fall 2007, paper: 07F-SIW-057.
3. IEEE Std. 1516.3™-2003, IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP).
4. IBM Rational Unified Process, http://www-306.ibm.com/software/awdtools/rup/
5. Hofmeister, C., Nord, R., ve Soni, D., Applied Software Architecture, Addison-Wesley Publisher, 2000, ISBN: 0-201-32571-3 .
6. The C4ISR Architecture Framework, http://www.c3i.osd.mil/org/cio/i3/default.htm
7. IEEE Std. 1471™-2000, Recommended Practice for Architectural Description of Software-Intensive Systems.
8. Clemens, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. ve Stafford, J., Documenting Software Architectures: Views and Beyond, Ed., Addison Wesley Publisher, 2002, ISBN: 0-201-70372-6.
9. Clemens, P., Kazman, R. ve Klein, M., Evaluating Software Architectures, Ed., Addison Wesley Publisher, 2002, ISBN: 0-201-70482-X.

Komuta Kontrol Yazılımlarında Mimari Tasarım Yöntemi Uygulama Deneyimi

Ertuğrul Barak[1], Sezen Erdem[2], Orçun Dayıbaş[3], Tankut Koray[4]

[1,2,3,4] Aselsan A.Ş. SST-MD-YMM, 06172, Yenimahalle, Ankara
[1] ebarak@aselsan.com.tr, [2]erdem@aselsan.com.tr, [3] odayibas@aselsan.com.tr,
[4]tkoray@aselsan.com.tr

Özet. Yazılım projelerinde, iyi mimari tasarımların karmaşıklıkla baş etmede ve kaliteli ürünler çıkarmada büyük bir rolü olduğu yadsınamaz bir gerçektir. Bu nedenle iyi yazılım mimarileri oluşturmak için uygulanacak yöntemler konusunda hem sanayide hem de akademik dünyada değişik fikirler ortaya atılmakta ve yöntemler önerilmektedir. Bu makalede bir komuta kontrol yazılımı projesi olan Yeni Nesil Teknik Ateş İdaresi projesinde bu yöntemlerden birinin uygulanış deneyimi aktarılmaktadır. Makalede uygulanan mimari tasarım yönteminin anlatımının yanı sıra, mimari tasarım yönteminin projede ne gibi değişikliklerle uygulandığı ve bu uygulama ile elde edilen tecrübeler ve kazanımlar anlatılmaktadır.

1 Giriş

Yazılım geliştirme dünyasındaki "yazılım krizi" olarak adlandırılan arz talep dengesizliği, konu ile ilgili kişileri sistemlerin karmaşıklıklarını nasıl daha kolay ele alabileceklerini araştırmaya itmiştir. Diğer bir yandan da ihtiyaç duyulan sistemlerin karmaşıklık ve büyüklükleri artmış, geleneksel algoritma ve veri yapıları tasarım için yetersiz kalmaya başlamıştır [1]. Bu iki konuya çözüm olarak ortaya çıkan yazılım mimarisi kavramı, yazılım bileşenlerine daha üst seviye bir soyutlama düzeyi ile yaklaşma fikrini temel alır. UML'in geliştiricilerinden Booch'un "tüm yazılım mühendisliği tarihi soyutlama düzeyinin artırılmasından ibarettir" görüşü [2] de bu yaklaşımı destekler niteliktedir.

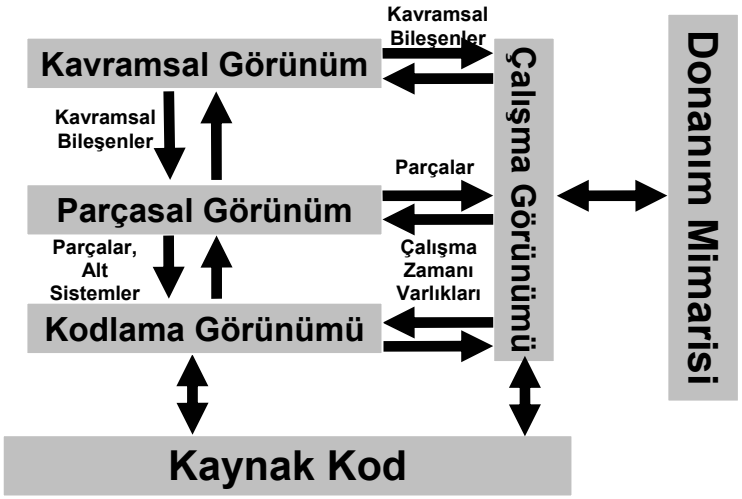
Bir bilişim sistemi ya da uygulamanın yazılım mimarisi, yazılım bileşenlerinden oluşan sistem yapısını/yapılarını, bu bileşenlerin dış özelliklerini ve birbirleri arasındaki ilişkileri içerir [3]. Yazılım mimarisi sistem gereksinimleri ile sistem gerçekleştirimi arasında bir köprü olarak değerlendirilebilir. Bu bağlamda mimari, kabaca sistemin sınırlarını çizerek temel çözümleme ve kararlara ışık tutabilen bir bilgi kaynağıdır. Örneğin, yeterli soyutlama düzeyinde ifade edilmiş yazılım mimarisi, sistemin farklı paydaşlar tarafından anlaşılması amacı ile kullanılabilir. Mimarinin ifade edileceği soyutlama düzeyi kullanım amacı ile doğrudan ilintili olduğundan, basit olmayan bir yazılım mimarisi farklı amaçlar (çözümleme, gerçekleştirim, yeniden kullanım, …) için farklı şekillerde ifade edilmelidir. Yine aynı bağlamda, aynı mimarinin farklı bakış açılarından farklı görünümleri olduğu söylenebilir.

Mimari tasarım yöntemleri ile ilgili pek çok yaklaşım, endüstriyel ya da akademik ortamda varlığını sürdürmektedir. Endüstriyel ve akademik kişilerden oluşan bir ekip

tarafından hazırlanan beş yazılım mimari yaklaşımının (ADD, S4V, RUP 4+1, BAPO, ASC) karşılaştırmalı incelemesinin yer aldığı bildiri [4], bu yöntemler hakkında giriş düzeyinde bilgi ve karşılaştırma içermektedir. Bu bildiri kapsamında özetlenen çalışma için Siemens Dört Görünüm(S4G) Yöntemi, benimsenmiştir.

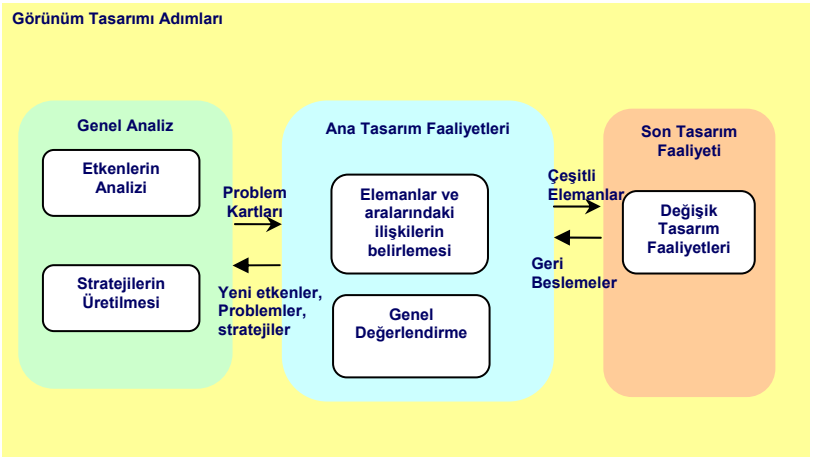
2 S4G Mimari Tasarım Yöntemi

Bu makalede adı geçen Siemens 4 Görünüm(S4G) mimari tasarım yöntemi Siemens Corporate Research tarafından, Siemens'te geliştirilen irili ufaklı pek çok karmaşık endüstriyel sistemin yazılım mimarileri üzerine yapılan araştırmalar sonucunda ortaya konan mimari tasarım yöntemidir[5]. Adından da anlaşılacağı üzere yazılım mimarisini 4 farklı görünüm ile ifade etmeyi öngörür. Yöntem, bu 4 görünümün nasıl bir akış içerisinde oluşturulacağını anlatmakta ve her bir görünümün oluşturulmasını kolaylaştıracak bazı yönlendirmeleri içermektedir. Yöntemin ayrıntılarıyla anlatıldığı ve uygulama örneklerinin de verildiği [5]'de yönteme adını veren görünümler Şekil 1'deki gibi ifade edilmektedir.



Şekil 1. S4G Mimari Tasarım Yöntemi Tasarım Görünümleri

Yöntem, bu görünümlerin, tekrar eden genel analiz işlemi çerçevesinde belli bir sırayla oluşturulmasına dayanmaktadır[4]. Bütün görünümlerin tasarım adımları, kendine özgü tasarım işlemleri gerektirse de temelde Şekil 2'de verildiği gibi yapılandırılmıştır.



Şekil 2. S4G Görünüm Tasarımı Adımları

Önce genel bir analiz aşaması olur ve bu aşamada mimariyi etkileyecek harici etkenler ve gerekler belirlenir. Bu gerekler üzerinde analizler yapılır. Bu etkenler ya da gereklerin tamamı sağlanamıyorsa gerçeklenebilir bir mimari oluşturulmasını sağlamak için gereklerin önceliklendirilmesi, gereğin değiştirilmesi ya da bazı dış etkenlerin değiştirilmesi gibi stratejilerin geliştirilmesine çalışılır.

İkinci olarak ana(central) tasarım faaliyeti gerçekleştirilir ki burada tasarlanmaktaki olan görünüme göre çeşitli tasarım elemanları, elemanlar arasındaki ilişkiler ve bu elemanların nasıl yapılandırılabileceği belirlenir. Ana tasarım faaliyetinin bir diğer alt adımı da çoğunlukla ana tasarım faaliyetindeki kararların birbirlerine ve önceki görünüm tasarımlarında alınan kararlara etkilerinin değerlendirildiği genel değerlendirme adımıdır.

Görünüm tasarım adımlarının üçüncüsü de diğer tasarım işlemlerine biraz daha az bağlılığı olan son tasarım faaliyetidir. Bu adımda genelde kaynak bütçeleme, arayüz tasarımları gibi faaliyetler gerçekleştirilir.

Yöntemde adı geçen 4 görünümün tasarlanması, kavramsal tasarım, parçasal tasarım, çalışma tasarımı ve kodlama tasarımı sırasıyla gerçekleştirilir. Burada adı geçen tasarım görünümleri şu şekilde ifade edilebilir;

Kavramsal Tasarım. Sistemin, işlevleri gerçekleştiren kavramsal bileşenler ile bu bileşenler arasında veri değişimini ve eşgüdümü sağlayan kavramsal bağlantılara eşlendiği tasarım aşamasıdır.

Parçasal Tasarım. Kavramsal bileşen ve bağlantıların alt sistem ve parçalara eşlendiği ve bu alt sistem ve parçaların da çeşitli katmanlara yerleştirildiği tasarımdır. Bu tasarımda kavramsal çözümün mevcut yazılım platformları ve teknolojileri ile nasıl gerçekleştirileceği verilmeye çalışılır.

Çalışma Tasarımı. Sistemin hafıza kullanımı ya da donanım atamaları gibi çalışma zamanı elemanları ve onların niteliklerini tanımlar.

Kodlama Tasarımı. Bu tasarımda çalışma tasarımındaki çalışma zamanı elemanlarının nasıl dağıtım elemanlarına eşleneceği, parçasal tasarımda belirlenen parçaların nasıl kaynak kod elemanlarına eşleneceği ve dağıtım elemanlarının, kaynak kod parçalarından nasıl oluşturulabileceği tanımlanır.

3 Yeni Nesil Teknik Ateş İdaresi Projesi

Yeni Nesil Teknik Ateş İdaresi Projesi (YNTAİ), Türk Ordu'sunun ateş destek sistemleri ihtiyacını taşınabilir modern cihazlar ve performansı yüksek yazılımlar ile çözmeyi hedefleyen bir projedir. Proje kapsamındaki birimler, kolay taşınabilir olmaları nedeniyle çok küçük taşınabilir kişisel bilgisayarlar ve el bilgisayarları olarak seçilmişlerdir.

YNTAİ yazılımları, bu bilgisayarların özellikleri göz önünde bulundurularak geliştirilmiştir. Çok küçük taşınabilir kişisel bilgisayarlarda masaüstü bilgisayarlarda kullanılan bir işletim sistemi çalıştırılabilirken, el bilgisayarlarında mobil cihazlar için özel olarak geliştirilmiş bir işletim sistemi kullanılmıştır. Benzer şekilde el bilgisayarları için özel olarak geliştirilmiş yazılım çerçeveleri kullanılmıştır.

Yazılım ihtiyaçları göz önüne alındığında bu iki tip bilgisayarda koşacak farklı yazılımların ortak bileşenlere sahip olabilecekleri değerlendirilmiştir. Tekrar kullanılabilirlik ve el bilgisayarının kullanımı ile ortaya çıkan kısıtlayıcı durumlar, yazılım mimarisine, ileriki bölümlerde detaylı şekilde anlatıldığı üzere, önemli derece etki etmiştir.

4 S4G Mimari Tasarımının YNTAİ Projesinde Uygulanışı

S4G mimari tasarım yöntemi, 2. bölümde de anlatıldığı üzere genel analiz faaliyeti çerçevesinde 4 farklı görünümün tasarlanması ve her yeni görünüm tasarımında gerek duyulursa, geriye dönülerek görünüm tasarımlarının güncellenmesi faaliyetlerini içerir. YNTAİ projesinde S4G uygulanırken farklı bir yaklaşım tercih edilerek genel analiz aşamasının bir kez yapılmasına ve bu daha kapsamlı genel analiz çalışmasının sonunda bütün tasarım görünümlerinin bir defada oluşturulmasına karar verilmiştir. Bu yaklaşımın benimsenmesinin temel nedeni, uygulama alanına çok hâkim olunması ve geçmişte benzer ürünlerin geliştirilmiş olmasıdır. Ayrıca geliştirme takviminin sıkışık olması da bu kararın alınmasındaki diğer etkenlerden olmuştur. Bu kararla birlikte tasarım yönteminin genel analiz aşaması ile uygulanmasına başlanmıştır.

Genel analiz aşaması, geçmişte geliştirilen ürünlerden elde edilen tecrübeler ışığında, mimariyi etkileyecek gereksinimlerin belirlenmesi ve bunların hararetli tartışmalar sonucunda ortaya çıkan bazı tasarımsal çözümlere eşlenmesi şeklinde gerçekleşmiştir. Bu aşamada, mimari tasarım yönteminin etkenleri sınıflandırma ve etkenlerin esnekliği ve değişkenliğini analiz etme yaklaşımlarının çok yönlendirici olduğu görülmüştür. Tasarım yönteminde etkenler kurumsal, teknolojik ve ürünsel etkenler olmak üzere başlıca üç gruba ayrılmaktadır. Özellikle bu gruplamanın etkenleri tespit etme ve tanımlamada büyük kolaylık sağladığı görülmüştür. Genel analizde yaşanan tasarım

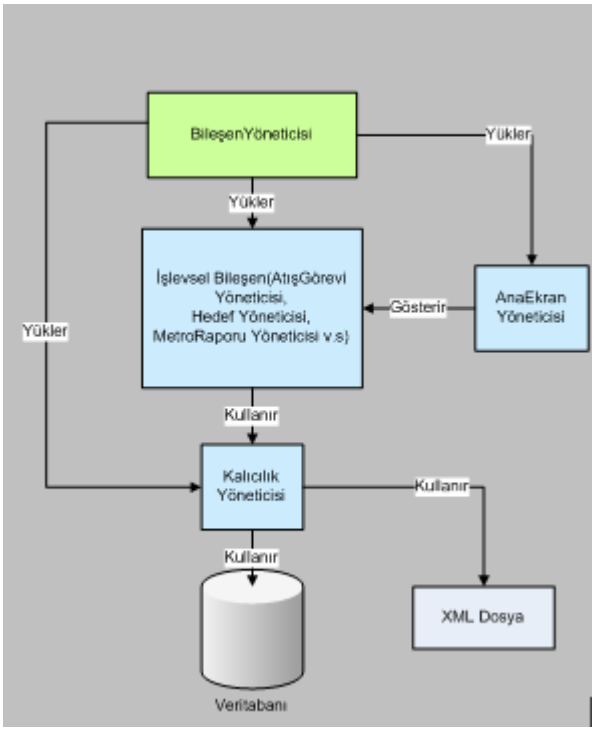
tartışmaları sonrasında ortaya çıkan küçük tasarım çözümleriyse, yöntemde adı geçen stratejileri oluşturmuştur. Genel analiz aşamasının çıktısı olarak yukarıda adı geçen sınıflandırmayı da içerecek şekilde bir etkenler stratejiler tablosu oluşturulmuştur. Aşağıda YNTAİ projesindeki bazı etken ve stratejileri içeren bu tablonun küçük çapta bir örneği verilmektedir.

Tablo 1. YNTAİ Projesi Etken ve Strateji örnekleri

Nu	Etkenler	Stratejiler
	Kurumsal	
1	Geliştirme takviminin sıkışık olması	Tekrar kullanılabilirlik en üst seviyede olmalı
	Teknolojik	
2	Veri saklama şekli ve altyapısının değişme ihtimali(VTYS yazılımı değişebilir, VTYS kullanılabilir ya da XML dosyası v.s. de kullanılabilir)	Veritabanı yönetim sistemi ya da XML dosyaları gibi farklı teknolojik alt yapıların kullanılmasına yönelik işlemler ayrı ve ortak bir arayüz üzerinden gerçekleştirilecek ve böylece veri saklama altyapısından bağımsızlık sağlanacaktır
	Ürünsel	
3	Donanım, işletim sistemi ve geliştirme çerçeve tasarımları bakımından iki farklı platform olması ve zaman içerisinde bunlara yenilerinin eklenme ihtimali	Kullanıcı arayüzleri, iş mantığı ve veri saklama işlemleri birbirinden ayrılacaktır İş mantığının olduğu bileşenlerde farklı çerçeve tasarımlara özel yapılar kullanılmyacaktır.

Genel analiz aşaması süresince daha önce farklı yazılımları geliştiren kişilerin etkenleri ve stratejileri tartışması, farklı bakış açılarını ve tecrübeleri paylaşmak adına bir fırsat oluştururken bir yandan da farklı bilgi ve tecrübe seviyesindeki insanların birbirlerini eğittiği bir faaliyete imkan vermiştir. Bu faaliyet sırasında, tüm mimariye odaklanılmayıp etkenler düzeyinde parça parça tasarım problemlerine yoğunlaşılsa da, herkesin kafasında bütünün resmi yavaş yavaş oluşmuştur. Bu aşamadan sonra da bu resmin oluşturulması ve paylaşılması gerçekleştirilmiştir. Bu da tasarım yönteminin ilk görünüm tasarımına, yani "Kavramsal Görünüm Tasarımı"'na karşılık gelmektedir. Bu aşamada ortaya çıkan resim Şekil 3'de verildiği gibi oluşmuştur. Bu resim, genel analiz

aşaması için verdiğimiz Tablo 1'deki örnek etken ve stratejiler çerçevesinde oluşturulmuştur ve asıl Kavramsal Tasarım Görünümünün sadece küçük bir parçasını göstermektedir.

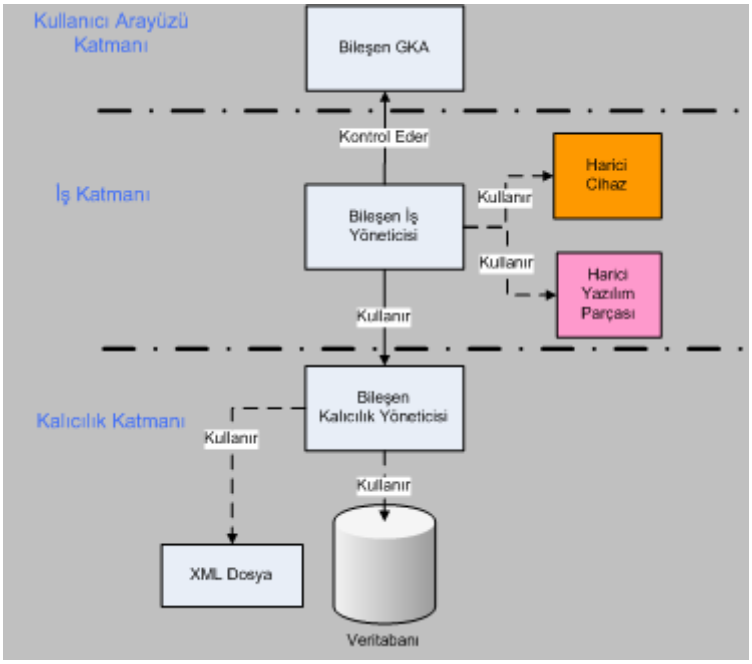


Şekil 3. YNTAİ Kavramsal Tasarım Görünümü Örneği

Kavramsal Görünüm Tasarımı'nda, ortaya çıkacak olan ürünün kavramsal olarak hangi parçalardan oluştuğunun gösterilmesinin yanında hangi kavramsal parçaların birbirleriyle ilişkilerinin olduğu da ifade edilmektedir. Tablo 1'deki etken ve stratejilerden 2 numaralı etkene yönelik geliştirilen stratejinin, yani veri saklama altyapısından bağımsızlığı sağlayacak bir bileşenin oluşturulması yaklaşımının ipuçlarını bu tasarım görünümünde bulmak mümkündür. Fakat 3 numaralı etkene yönelik uygulanacak stratejiyi bu görünümde hissetmek pek mümkün olmamaktadır. Bu stratejinin uygulanışını gözlemleyebileceğimiz görünüm mimari tasarım yönteminin öngördüğü ikinci tasarım görünümü olan Parçasal Görünüm Tasarımı'dır.

S4G tasarım yönteminde Parçasal Görünüm Tasarımının oluşturulması faaliyeti, kavramsal bileşen ve bağlayıcıların asıl kodu oluşturacak olan altsistemlere, parçalara ve arayüzlere dönüştürülmesi işlemini içerir. Ayrıca katmanlı bir yapıda çeşitli işlevsel katmanların belirlenmesi ve oluşturulan bu alt sistem ve parçaların belirlenen bu katmanlara atanması da bu aşamada yapılır. YNTAİ projesinde bu tasarım

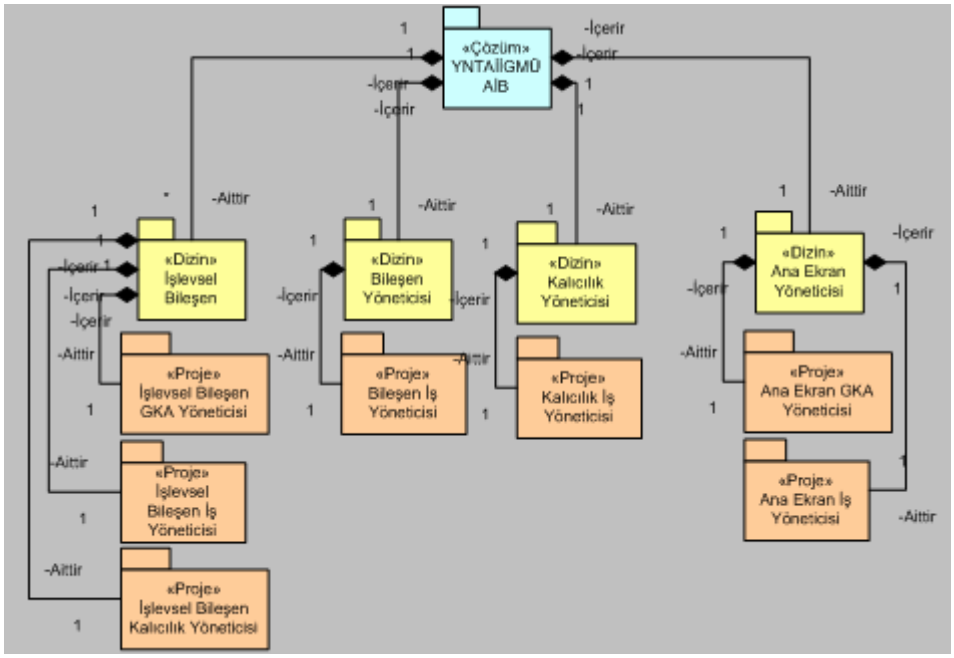
görünümünün oluşturulması sonrasında Şekil 4'deki gibi bir tasarım görünümü ortaya çıkmıştır. Bu şekilde, Şekil 3'te verilen Kavramsal Görünüm Tasarımında yer alan işlevsel bileşen kavramının hangi parçalara bölüneceği ve bu parçaların hangi katmanlara eşlendiği gösterilmektedir. Bu şekil yine bu makalede yer verilen örnek etken ve stratejiler çerçevesinde verilmektedir.



Şekil 4. YNTAİ Parçasal Tasarım Görünümü Örneği

S4G Mimari tasarım yönteminin bir sonraki adım olarak öngördüğü tasarım adımı Çalışma Görünümü'nün tasarlanmasıdır. YNTAİ'deki ürünler değerlendirildiğinde ve geçmiş tecrübelerimize de dayanarak bu görünümün tasarlanmasının çok kritik olmadığı değerlendirilmiş ve tasarlanmamasına karar verilmiştir. Bu kararın ileride kodlamaya geçildiği aşamada bazı tasarım düzeltmeleri gerektireceği öngörülememiştir. Bu konuya, edinilen kazanımlar ve uygulama sırasındaki gözlemler bölümünde değinilecektir.

S4G Mimari tasarım yönteminin önerdiği son tasarım görünümü Kodlama Görünümü Tasarımı'dır. Bu görünümde de kodlamanın nasıl gerçekleştirileceği fiziksel dosyalar bazında ifade edilmeye çalışılmakta ve kodlama ve dağıtım aşamaları disipline edilmeye çalışılmaktadır. Şekil 5'de görülen YNTAİ Kodlama Görünümü Tasarımı Tablo 1'deki örnek etken ve stratejiler çerçevesinde verilmektedir.



Şekil 5. YNTAİ Kodlama Tasarım Görünümü Örneği

YNTAİ'de yazılımların üzerinde koştuğu platformu donanım, işletim sistemi ve geliştirme çerçeve tasarımından oluşan üçlü bir grup olarak tanımlamak mümkündür. Böyle bakıldığında projedeki yazılımların iki farklı platform üzerinde çalışma gereksinimleri olduğu için yukarıda verilen kodlama tasarımı görünümünün her iki platform için ayrı ayrı oluşturulması gerekmiştir. Şekil 5'teki görünümde bu platformlardan sadece birinin üzerinde çalışacak olan, ileri gözetleyici ve ateş idare birimi yazılımlarının kaynak kodlarının ifade edildiği görünüm verilmektedir. İki platform için iki farklı görünüm oluşturulmuş olmasına karşın iş katmanı ve kalıcılık katmanında yer alan bileşenler tamamen ortaktır ve tek bir kez oluşturulup her iki çözüm tarafından da kullanılmıştır.

S4G mimari tasarım yönteminin YNTAİ projesindeki kullanımı sadece bu görünümlerin oluşturulmasından ibaret değildir. Özellikle Parçasal Görünüm Tasarımı aşamasında her bir kavramsal bileşenin alt parçalarının tanımlandığı, katmanlara ayrıldığı ve bu katmanlardaki parçaların arayüzlerinin bulunduğu diğer parçaların da gösterildiği görünümler çizilmiştir. Bu parçaların arayüzleri üzerinden ne gibi servislere ihtiyaç duyduğu ve diğer parçalara ne gibi servisler sağladığı da ayrıntılı bir şekilde belgelenmiştir. Makalenin bu bölümünde sadece buraya kadar belirtilen ayrıntılara yer verilecek ve bundan sonraki bölümde ağırlıkla edinilen kazanım ve tecrübelere değinilecektir.

5 Mimari Tasarım Yöntemi Kullanımı ile Edinilen Kazanımlar ve Uygulama Sırasındaki Gözlemler

Mimari tasarım yönteminin projede uygulanması ile öncelikle çok daha kaliteli ve esnek mimarilere sahip yazılımlar ortaya çıkmıştır.

Bu yazılımlarda donanım ve çalışma ortamlarının farklılığına rağmen yeniden kullanım seviyesi oldukça yüksek düzeyde gerçekleşmiştir.

Ana tasarımın ve gerçekleme yaklaşımlarının ortaya konması mimariyi oluşturan yapıtaşlarının standart bir şekilde gerçeklenmesini sağlamıştır.

Kodlama görünümünün kodlamaya geçilmeden önce ortaya konması gerçeklemenin düzenli bir şekilde olmasını sağlamıştır.

Ekibin tüm tasarıma katkısı ve katılımı sağlandığı için ortaya çıkan mimari tasarım da herkes tarafından benimsenmiştir. Bu da gerçekleme aşamasına geçildiğinde tüm ekibin uyumlu bir şekilde çalışmasını sağlamıştır.

Mimari tasarım yapmak yazılım geliştirme probleminin çözümüne bir bütün olarak üst düzey bir bakış sağladığı için daha sonra çıkması muhtemel problemlerin çok önceden farkedilebilmesini sağlamıştır.

S4G mimari tasarım yönteminin etkenlerin esnekliğinin analizine yaptığı vurgu sayesinde tasarım sırasında pazarlık yapma alışkanlığının elde edilmesi sağlanmıştır. Böylece gereksiz karmaşıklıkların ortadan kaldırılması ve gerçeklenebilir mimariler oluşturulması kolaylaşmıştır.

Bu kazanımlarının yanında S4G mimari tasarım yönteminin uygulanması sırasında yöntem ve uygulama şeklinden kaynaklanan çeşitli ilginç durumlar gözlenmiştir. S4G mimari tasarım yönteminin YNTAİ projesi kapsamında biraz farklı uygulandığı daha önceki bölümlerde ifade edilmişti. Farklılık olarak da Genel Analiz aşamasının bir kez yapılması ve tüm tasarım görünümlerinin bundan sonra ortaya çıkarılması ifade edilmişti. Yöntemin bu şekilde uygulanması Genel Analiz aşamasının biraz uzun sürmesine ve sıkışık proje takvimi nedeniyle de yazılım ekibinde vakit kaybediyoruz endişelerinin oluşmasına neden olmuştur. Bu endişeler uygulamadan vazgeçme önerilerinin bile gündeme gelmesine neden olmuş, ama ısrarlı bir şekilde uygulamaya devam edilmesi çalışmanın başarıyla tamamlanmasını sağlamıştır.

Genel Analiz aşamasında, etkenlerin belirlenmesinden sonra stratejiler tüm ekibin katıldığı tasarım tartışmaları ile ortaya konmaya çalışılmıştır. Farklı bilgi seviyesi ve tecrübeden insanların bu toplantılarda bir arada olması ekip içinde bilgi ve tecrübenin paylaşılması açısından faydalı olmuş ve bir nevi ekip içi eğitim görevi görmüştür. Bu arada bazı tasarım toplantılarında zaman zaman şiddetli tartışmalar yaşanabilmiş ve farklı bilgi seviyesi ve tecrübeden insanların kendilerini baskın kılmaya çalıştıkları durumlar da gözlenmiştir. Bu tip toplantılarda verimin artabilmesi için daha çok katılımın sağlanması ve baskın karakterlerin toplantıları domine etmesine izin verilmemesi gerektiği gözlenmiştir.

S4G yöntemindeki Genel Analiz gibi bir faaliyetle daha en baştan kalite gereksinimlerine odaklanmanın daha sonradan çıkabilecek problemlere de çözümler

sunabildiği gözlemlenmiştir. YNTAİ'de yaşanan veritabanı yönetim sistemi problemi buna örnek olmuştur. YNTAİ projesinde yazılımların üzerinde koşacağı bir kısım donanımlar, yazılım geliştirme aşamasından önce temin edilememiştir. Geliştirme aşaması bu donanımlar üzerinde koşacağı varsayılan bir ticari VTYS yazılımı kullanılarak devam etmiş fakat donanımlar tedarik edildiğinde bu VTYS yazılımının gelen donanımlar tarafından desteklenmediği görülmüştür. Ancak uygulanan yöntem ile bu etken çok önceden ele alındığı ve esnek bir yazılım mimarisi oluşturulduğu için bu problem kolayca aşılmış ve yeni bir VTYS ile yola devam edilebilmiştir.

S4G yönteminin uygulanması sırasında Çalışma Görünümü tasarımının kritik olmadığı değerlendirilerek oluşturulmamasına karar verilmiştir. Bu kararın alınması çalışma zamanındaki iş parçaları yönetimine yeterince odaklanılamamasına neden olmuş ve geliştirme sırasında bazı problemlerle karşılaşılmıştır. Özellikle kullanılan çerçeve tasarımından kaynaklanan kullanıcı arayüzlerinin işçi iş parçaları ile güncellenmesi problemleri yaşanmıştır. Bu da ayrıntılı tasarımda önemli değişiklikler gerektirmiştir.

6 Sonuç

Bu makalede endüstri tarafından ortaya atılan bir mimari tasarım yöntemi olan Siemens 4 Görünüm Mimari Tasarım Yönteminin bir Komuta Kontrol Yazılımı projesi olan Yeni Nesil Teknik Ateş İdaresi Projesi'nde ne şekilde uygulandığı ve bu uygulamadan elde edilen sonuçlar anlatılmıştır. Uygulamadan oldukça başarılı sonuçlar elde edilmiş, kısa süre içerisinde iki farklı platform üzerinde çalışan üç ayrı yazılım yüksek bir yeniden kullanılabilirlik seviyesi ile üretilebilmiştir. Bu da olgun bir mimari tasarım yöntemi kullanmanın yazılım projelerinin başarısı açısından ne kadar önemli olduğunu göstermektedir.

Kaynakça

1. David Garlan, Mary Shaw, An Introduction to Software Architecture, 1994.
2. G. Booch, (2000), Handbook of Software Architecture, www.booch.com/architecture.
3. L. Bass, P. Clements, R. Kazman, "Software Architecture in Practice", Addison Wesley, 2003 ISBN: 0321154959.
4. C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, P. America, Generalizing a Model of Software Architecture Design from Five Industrial Approaches, ,WICSA5, 2005.
5. C. Hofmeister, R. Nord, and D. Soni, Applied Software Architecture, Boston: Addison-Wesley, 2000 ISBN: 0201325713.

Gereksinim Güdümlü Uygulama Geliştirme Çerçevesi "ReDSeeDS" ile Bir Uygulama

Özgür Tüfekçi[1], İlker Çokkeçeci[2] , Semih Çetin[3]

[1,2,3] Cybersoft C/S Enformasyon Teknolojileri, ODTÜ Teknokent Silikon Blok No:18, Ankara
{ozgur.tufekci[1], ilker.cokkececi[2], semih.cetin[3]}@cs.com.tr

Özet. ReDSeeDS; Avrupa Birliği 6. Çerçeve Programı'nda yer alan ve gereksinim güdümlü yazılım geliştirme yaklaşımını benimseyen bir çerçeve ile buna uygun bir yazılım geliştirme metodolojisi ortaya koymayı amaçlamaktadır. ReDSeeDS (Requirement-Driven Software Development System) yaklaşımı; kullanıcı senaryolarına dayanan, model güdümlü mimari üzerine inşa edilmiş ve yeniden kullanımı en üst düzeye taşımayı hedeflemektedir. Birbirinden soyutlanabilen "yazılım durumları (software cases)" ile "Gereksinimlerden => Mimariye", "Mimariden => Tasarıma" ve "Tasarımdan => Koda" geçişi sağlayabilecek dönüşümler, ReDSeeDS araç seti tarafından otomatik ve/veya yarı-otomatik olarak gerçekleştirilmektedir. ReDSeeDS, temel hatlarıyla "durum temelli çıkarım (case-based reasoning)" yaklaşımını; gereksinimlerden koda çevirebilecek şekilde ele alan, kapsamlı bir süreç sunmaktadır. Bildiride ReDSeeDS yaklaşımının kısaca aktarılmasının ardından, projede yer alan 11 proje ortağından birisi olan Cybersoft'un bu proje kapsamında yaptıkları açıklanmakta, deneyimleri paylaşılmakta, projenin bu aşamaya kadar geldiği durum değerlendirilmekte ve bundan sonraki süreçte gerçekleştirilecekler aktarılmaktadır.

1 Giriş

Yazılım mühendisliği, yazılım geliştirme sürecinin kolaylaştırılması, sistem ve kullanıcı gereksinimlerinin karşılanması, bakım ve yönetiminin azaltılması ile tüm bunların daha ucuz bir şekilde sağlanması adına birçok yöntem ve araç ortaya koymuştur. Geriye bakıldığında bu amaçların birçoğuna ulaşılmış olunmasına karşın, gereksinimlerin tam olarak karşılanabilmesi ve bu karşılamanın fiyat-etken bir şekilde sonuçlandırılabilmesi konularında hala daha açık noktalar olduğu görülmektedir. Özellikle işlevsel olmayan gereksinimlerin (NFR: Non-Functional Requirements) belirlenmesi, modellenmesi, işlevsel gereksinimlerle örtüştürülmesi ve tüm bunların yazılım geliştirme süreci boyunca izlenebilmesi yazılım mühendisliği araştırmacılarını hala meşgul eden konular arasındadır.

Avrupa Birliği 6. Çerçeve Programı kapsamında sürdürülmekte olan ReDSeeDS projesi [1]; model güdümlü bir uygulama geliştirme yaklaşımı ile gereksinimleri mimariye, mimariyi tasarıma ve tasarımı koda bağlayarak, birbirinden soyutlanabilen "yazılım durumları" aracılığıyla yeniden kullanımı en üst düzeye taşımayı amaçlayan bir yazılım geliştirme süreci ve araç kümesi önermektedir. ReDSeeDS projesinde yer alan 7 Avrupa ülkesinden 11 katılımcının ortak hedefi; gereksinim mühendisliği alanındaki mevcut birikimden yola çıkarak meta modelleme, model dönüşümü, sorgulama ve çıkarım

teknikleri ile durum temelli yeniden kullanılabilir yazılım geliştirme için tamamen yeni bir yaklaşım altyapısı oluşturabilmektir.

ReDSeeDS; yazılım mühendisliğinde "durum temelli çıkarım (case-based reasoning)" yaklaşımını gereksinimlerden koda çevirecek şekilde ele alan, oldukça geniş kapsamlı bir süreç sunmaktadır. Bu amaçla öncelikle işlevsel ve işlevsel olmayan gereksinimlerin "SVO(O) Subject-Verb-Object-(Secondary Object)" formatında ifade edilmesi ile bu ifadelerin işlenmesi sonucu bir veritabanında (repository) saklanması, sonrasında Model Güdümlü Geliştirme yaklaşımını kullanılarak, ilgili ReDSeeDS araçları yardımıyla bu gereksinimlerden mimari, tasarım ve kod parçacıklarının oluşturulması sağlanmaktadır.

Cybersoft; ReDSeeDS projesinde son ürünün kullanımı için gerekli belirtimlerden, bunların sınanmasından ve yaklaşımın pratik kullanımları sonucu proje ortaklarına geri beslemenin sağlanmasından sorumlu üç uzman kurumdan birisidir. Bildiride ReDSeeDS yaklaşımının kısaca sunulmasının ardından, projede yer alan 11 proje ortağından birisi olan Cybersoft'un proje kapsamında gerçekleştirdikleri açıklanmakta, ReDSeeDS yaklaşımı, metodolojisi ve araçları ile yazılım geliştirme pratikleri Cybersoft'un yaşadığı deneyim sonrası değerlendirilmektedir. Bildiri; sonuçların ortaya konulması ve ReDSeeDS projesi bünyesinde bundan sonraki dönemde gerçekleştirilecek aktivitelerin kısaca aktarılması ile son bulmaktadır.

2 Problem Tanımı ve Motivasyon

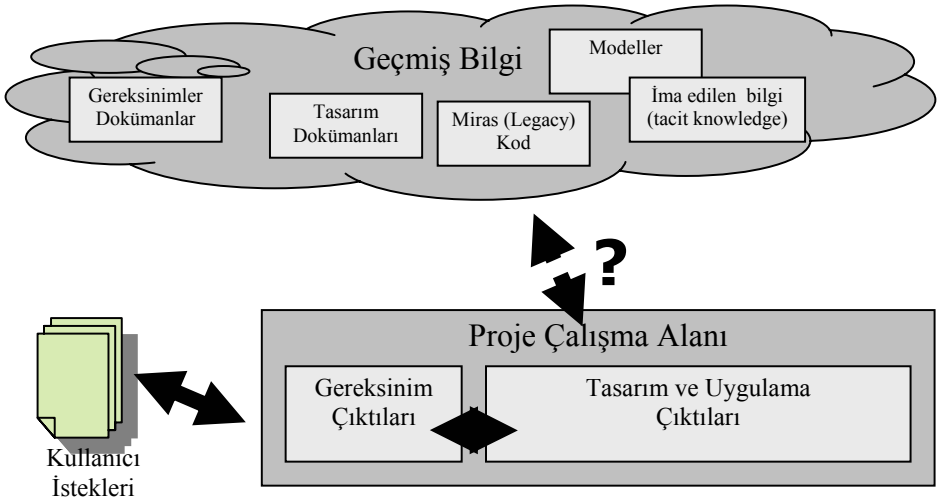
Donanım endüstrisindeki hızlı gelişmeye ayak uyduramayan yazılım sektörü, kullanıcı gereksinimlerinde artan karmaşıklık seviyesini yönetmekte ve azalan pazarlama zamanı kısıtlarını karşılamakta darboğazlar yaşamaktadır. Bununla birlikte yazılım geliştiriciler çoğu zaman bir yazılım geliştirme sürecinde yaptıkları işleri daha önce başka bir yerde aynen veya çok benzer bir şekilde gerçekleştirdikleri hissiyatına kapılırlar. Bu olmasa bile bir yazılımcının mimari, teknolojik altyapı, tasarım örüntüleri vb. kapsamında mevcut gerçekleştirdiği aktivite/aktivitelerin çok benzer durumlar için diğer ekip arkadaşları tarafından daha önce gerçekleştirilmiş olma olasılığı da oldukça fazladır. Önemli olan, yazılım geliştirme süreci bünyesinde bu tür tekrarlanabilen kullanım durumlarının sınıflandırılabilmesi, sorgulanabilir hale getirilebilmesi ve bu durumlardan azami seviyede yeniden kullanım aşamasında yararlanılabilmesidir [8].

Yeniden kullanım çok uzun zamandır yazılım geliştirmede anahtar çözüm olarak görülse de, karmaşıklık ve geliştirilen yazılım sistemlerinin çokluğu sonucunda daha önce geliştirilen yazılım durumları dikkate alınmamaktadır. Bu durumun nedenleri arasında farklı yazılımcıların farklı projelerde çalışıyor olması, yazılım durumları hakkındaki bilgiyi paylaşamıyor olmaları, uzun geliştirme çevrimlerinde önceki çözümlerin unutuluyor olması ve önceki çözümlerle benzerliklerin bulunmasının zor olması sıralanabilir [2].

Yazılım projeleri bünyesinde gereksinim dokümanları, tasarım dokümanları, kod gibi yazılıma özgü bir takım çıktılar üretilmektedir. Ayrıca, yazılım ekipleri proje boyunca "bilindiği varsayılan bilgiyi (tacit knowledge)" edinmektedirler. Bazı proje çıktıları, form tasarımı ve analitik örüntülerle genelleştirilebilirse de ne yazık ki yeni karşılaşılan problemler bir öncekiyle çok fazla benzerlik gösterse dahi çıktıların yeniden kullanımı genellikle çok zordur. Bu zorluğa sebep olan neden, mevcut bilginin kolay karşılaştırma

ve tekrar erişim için yapısal bir çözüm sunmamasıdır [3, 8]. Mevcut yazılım projelerinde bilgiyi yeniden kullanmanın yetersizliği Şekil 1'de verilmektedir.

Mevcut modelleme ve transformasyon dilleri bütünüyle yeniden kullanılabilir yazılım durumları tanımlama potansiyeline sahip değildir. Bu diller daha çok tasarım modellerine odaklanmakta ve farklı seviyelerdeki tasarım modelleri arasında transformasyon sağlamaktadır [3].



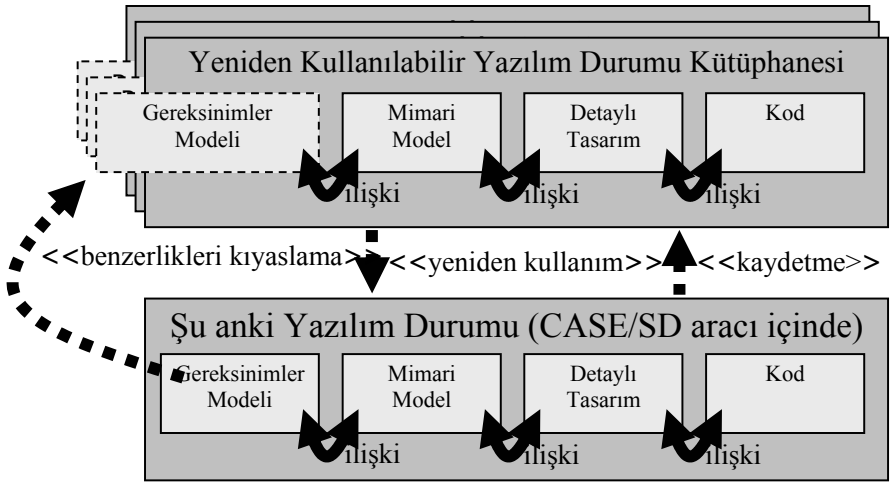
Şekil 1. Yazılım geliştirme projesinde yapısal olmayan bilginin yeniden kullanımı

Tasarım örüntüleri veya bileşen/uygulama çerçeveleri gibi yazılımın yeniden kullanımı için geçerli mevcut yaklaşımların getirileri sınırlı olmakta ve bu yaklaşımlar "yeniden kullanımla birlikte geliştirme" yerine "yeniden kullanım için geliştirme"yi öngördükleri için yeniden kullanım mekanizmasının hayata geçirilmesi kayda değer bilgi ve emek gerektirmektedir. Diğer bir yaklaşım olan yazılım ürün hatlarında ise yeniden kullanım için başlangıçta yüksek emek ihtiyacı meydana gelmektedir [2]. Yeniden kullanımın etkinliği ve verimliliğini artırabilmek için yeniden kullanılacak ürünlerin oluşturulmasını ve kullanımını basitleştirip, elverişli hale getirecek yaklaşımlara ihtiyaç duyulmaktadır.

3 Çözüm Önerisi: ReDSeeDS

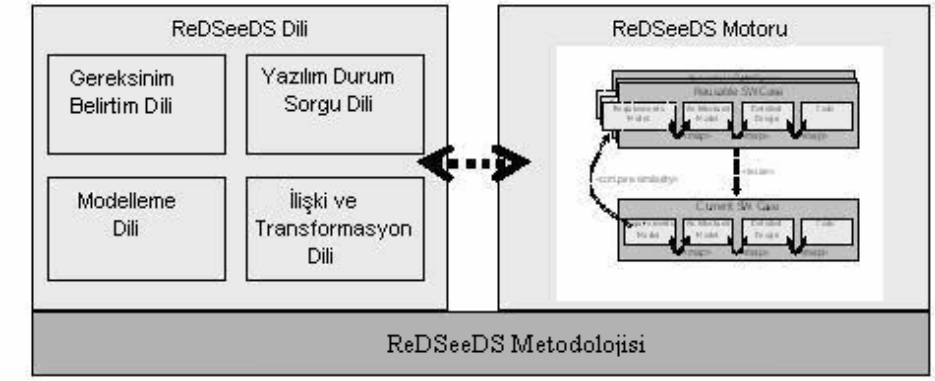
ReDSeeDS'in ana fikrini anlamak için "yazılım durumu" kavramını tekrar tanımlamak gerekir. Yazılım durumu tanımı; gereksinim modeli, mimari model, detaylı tasarım ve kodu içermektedir [4]. Şekil 2'den anlaşılacağı üzere gereksinim modeli; problemi tarif etmekte, diğer yandan mimari model, detaylı tasarım ve kod ile birlikte çözümü göstermektedir. Bu şekilde genişletilmiş olan "yazılım durumu" kavramı aslında

yazılım gereksinim modeli formunda tam olarak ifade edilmiş problem tanımını adreslemektedir. Problem tanımının tüm öğeleri daha sonra çözümün ilgili öğeleriyle ilişkilendirilmektedir. Çözüm, tasarım modeli (mimari model, tasarım model vb. içeren) ve nihai kodu kapsar. Yazılım durumları halen geliştirilmekte olan sistemlerdeki (şu andaki yazılım durumu) benzerlikler temel alınarak yeniden kullanılabilir. Bu benzerlik şu andaki (halen bitmemiş olabilir) gereksinim modeli ile geçmiş yazılım durumu gereksinim modellerinin <<karşılaştırılmasıyla>> belirlenebilir. Önceki çözüm, işaretlenen yerlerin yeniden ele alınmasıyla şu anki problemi çözmek için <<yeniden kullanılabilir>> [3]. ReDSeeDS yaklaşımı, buna yönelik olarak "yazılım durumu" tanımlamalarının "belli bir meta-modele uygun modeller" ile "belli bir gramere dayalı formal lisanlarla ifade edilebilen belirtimler" şeklinde gösterimini öngörmektedir.



Şekil 2. Gereksinim güdümlü yapısal yazılım bilgisi tekrar kullanımı

Bir uygulama alanına ait bütün yazılım durumları, yazılım bilgi modeli içinde toplanıp özetlenir. Yazılım bilgi modelindeki değişkenlik; yazılım geliştirilmesi sırasında şu şekilde kullanılır: gereksinim belirtimleri temel alınarak, benzer yazılım durumları (diğer bir ifadeyle benzer isterlere sahip yazılım durumları) belirlenir. Erişilen yazılım durumları aynı zamanda "mimari, detaylı tasarım ve kod çözümü" şeklinde ifade edilir. Gereksinimler, mimari, detaylı tasarım ve kaynak kodu arasındaki izlenebilirlik ilişkileri sayesinde tekrar kullanım ve/veya uyarlanmış (kısmi) çözümler mümkün hale gelir [5]. Bu vizyon ile proje ReDSeeDS kapsamlı yeniden kullanılabilirliği sağlayan, gereksinim güdümlü detayları Şekil 3'te verilen bir uygulama geliştirme çerçevesi sunmaktadır.



Şekil 3. ReDSeeDS Çerçevesinin öğeleri

Bu çerçeve ReDSeeDS dili, ReDSeeDS motoru ile ilintili ReDSeeDS metodolojisinden oluşmaktadır. Yeniden kullanılabilir yazılım durumlarının tanımlandığı ReDSeeDS dili de Gereksinim Belirtim Dili (Requirements Specification Language), Modelleme Dili (Modeling Language), İlişki ve Transformasyon Dili (Model Transformation Language) ile Yazılım Durum Sorgu Dili (Software Case Query Language)'nden oluşmaktadır. ReDSeeDS Metodolojisi, ReDSeeDS Motoru vasıtası ile yazılım durumlarının yaratılmasını ve yeniden kullanılmasını içeren bir yazılım yaşam döngüsü yönetim yaklaşımı sunmaktadır. ReDSeeDS Metodolojisi yazılım geliştirme sürecini değiştirerek yeniden kullanıma hazırlık ihtiyacını ortadan kaldırmaktadır.

4 Cybersoft'un ReDSeeDS Deneyimleri

Cybersoft'un proje içerisindeki temel rolü kullanıcı olarak ReDSeeDS çerçevesinin endüstriyel uygulamadaki geçerliliğini sınamaktır. Bu aşama, kullanışlı ve tekrar kullanılabilen bir platform oluşturmak adına çok önemlidir. Bunun için gerçek yaşamdan bir proje üzerinde karşılaştırmalı bir çalışma yapılmaktadır. Cybersoft'un Aurora Yazılım Üretim Bandı [6, 9] ile geliştirdiği, Web tabanlı, 4 katmanlı mimari üzerinde merkezi bir yapıda çalışan ve 3000'in üzerinde kullanıcı tarafından kullanılan Tedaş (Türkiye Elektrik Dağıtım A.Ş.) Bilgi Yöntem Sistemi (BYS) Projesi Stok Kontrol Sistemi uygulamasından faydalanılmaktadır. Çalışma adımları ilerleyen bölümlerde örneklerle verilmektedir. Yer darlığı yüzünden, örnekler sınırlı tutulmuştur.

4.1 Gereksinim Belirtim Dilinin (RSL) Uygulanabilirliğinin Sınanması

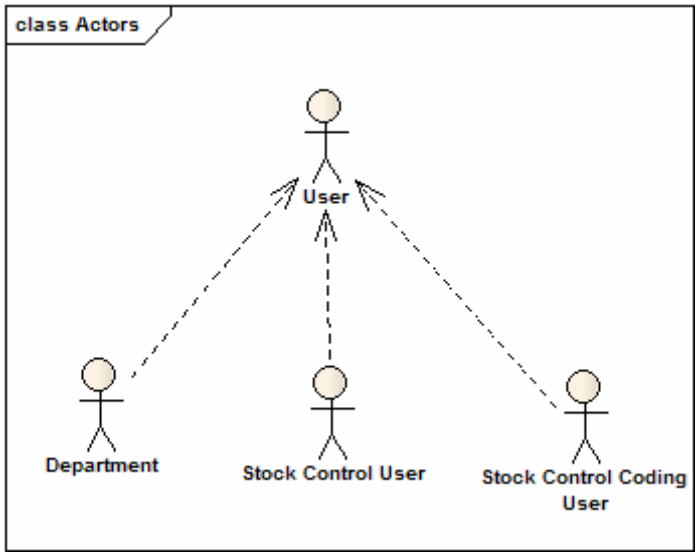
Bu çalışmanın amacı tutarlı, doğru ve anlaşılabilir bir gereksinim belirtim dilinin oluşturulmasına katkı sağlamaktır. Stok Kontrol Sistemindeki işlevsel kullanıcı gereksinimleri Gereksinim Belirtme Dili (RSL) kullanarak aşağıda örneklendiği şekilde belirtilir.

İşlevsel gereksinimlerin SVO(Subject-Verb-Object) formatında ifade edilmesi:

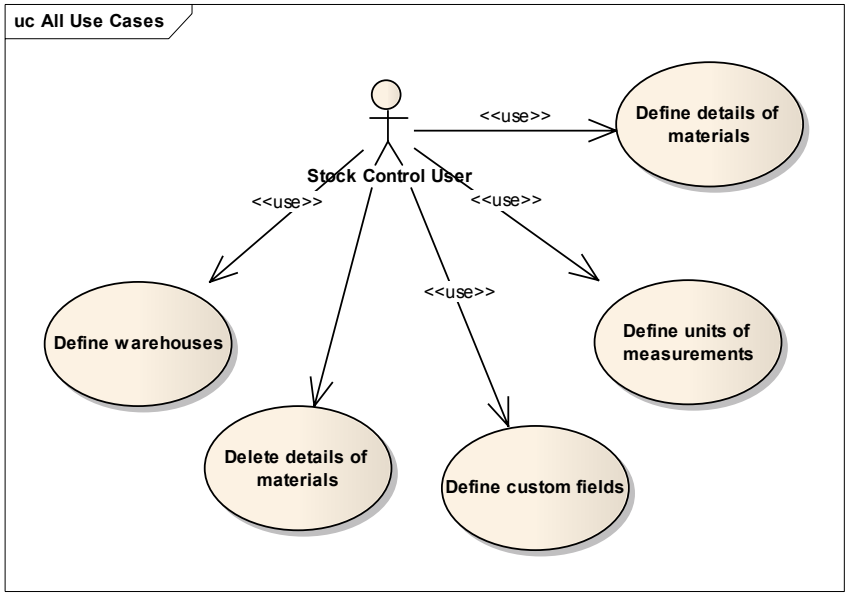
1.[s: Stok Kontrol Kullanıcısı] [v: tanımlar] [o: ambar].
2.[s: Stok Kontrol Kullanıcısı] [v: tanımlar] [o: özel_alan].
3.[s: Stok Kontrol Kullanıcısı] [v: tanımlar] [o: ölçüm-birimi].
4.[s: Stok Kontrol Kullanıcısı] [v:siler] [o: malzeme-bilgileri].
5.[s: Stok Kontrol Kullanıcısı] [v:kayıt eder] [o: malzeme-bilgileri].

4.2 Transformasyon Dilinin Sınanması

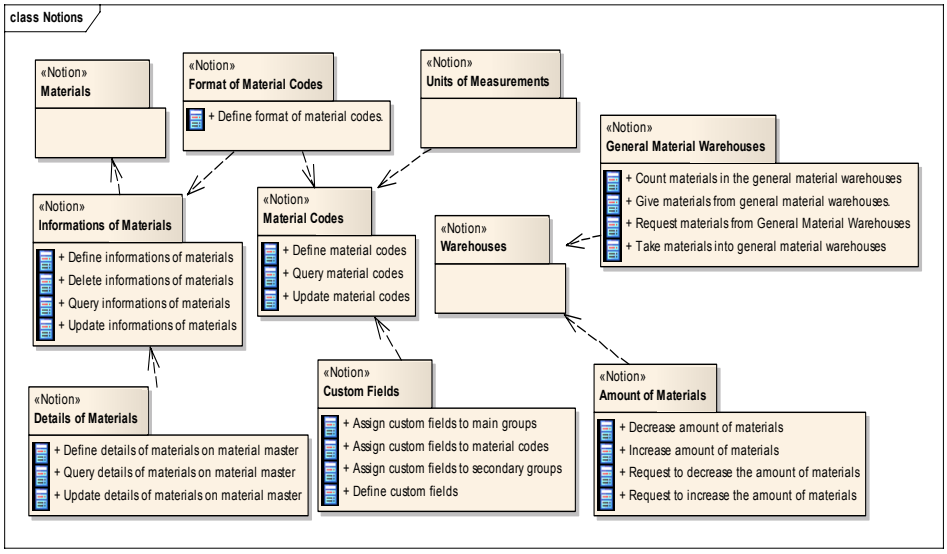
Transformasyon dilinin, transformasyon kuralları ve geliştirilmiş modeller arasındaki uygulanabilirliği Stok Kontrol Sistemi uygulaması ile sınanmıştır. Bu aşamada ReDSeeDS modelleme aracı kullanılarak, Aktörler (Şekil 4) ve Sistem Elemanları tanımlanır. SVO formatında ifade edilen işlevsel gereksinimlerden Kullanım Durumları (Şekil 5) çıkartılır ve "Nosyon" olarak nitelendirilen Alan Elemanları ve Alan Tanımlaması (Şekil 6) yapılır. Bu işlemlerin temel amacı yeniden kullanılabilirliği de göz önünde tutan bir "alan modeli (domain model)" oluşturmaktadır. Yazılım ürün hatlarında genellikle kullanılan "özelliğe yönelik alan modeli (feature-oriented domain model)" yerine ReDSeeDS yaklaşımında tanımsal "SVO(O) ile" gereksinimlerin her aşamada takibinin (requirements traceability) yapılabildiği ve model güdümlü mimari ile çevrime tabi tutulabilecek bir alan modeli söz konusudur.



Şekil 4. Stok Kontrol Sistemi Aktörleri



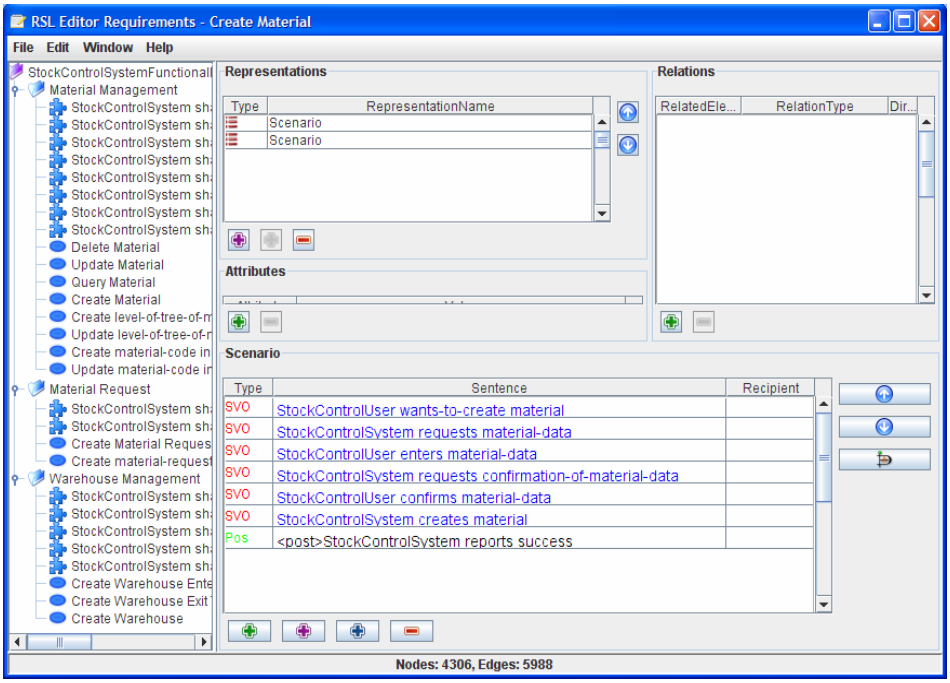
Şekil 5. Kullanım Durumları



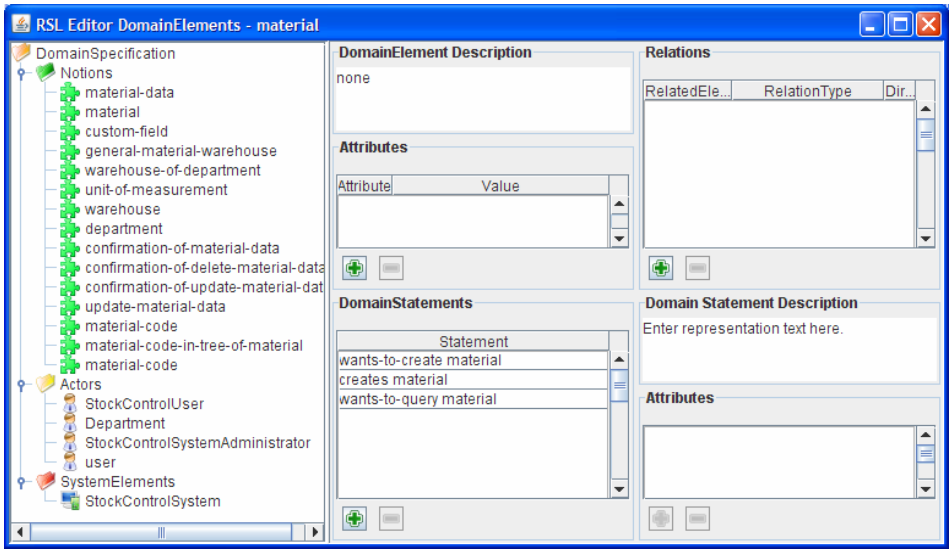
Şekil 6. Alan Elemanları ve Alan Tanımlaması

4.3 Gereksinimlerin RSL Editör'de Tanımlanması ve Transformasyon İşlemi Yapılarak Mimari Model Oluşturulması

İşlevsel gereksinimler; Kullanım Durumları ve Senaryolar Şekil 7'de görüldüğü gibi RSL Editör'de tanımlanır. Ayrıca Şekil 8'de belirtildiği gibi Alan Elemanları olarak Aktörler, Sistem Elemanları ve Nosyonlar işlevsel gereksinimler ile ilişkili olarak tanımlanmaktadır.

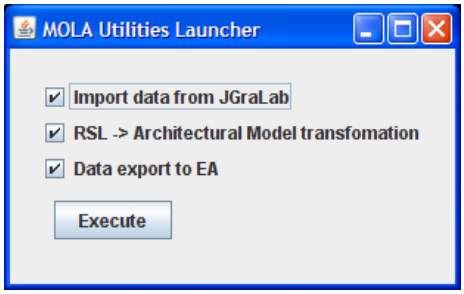


Şekil 7. İşlevsel gereksinimler, Kullanım Durumları ve Senaryoların RSL Editör'de tanımlanması



Şekil 8. Alan Elemanların işlevsel gereksinimler ile ilişkili olarak RSL Editör'de tanımlanması

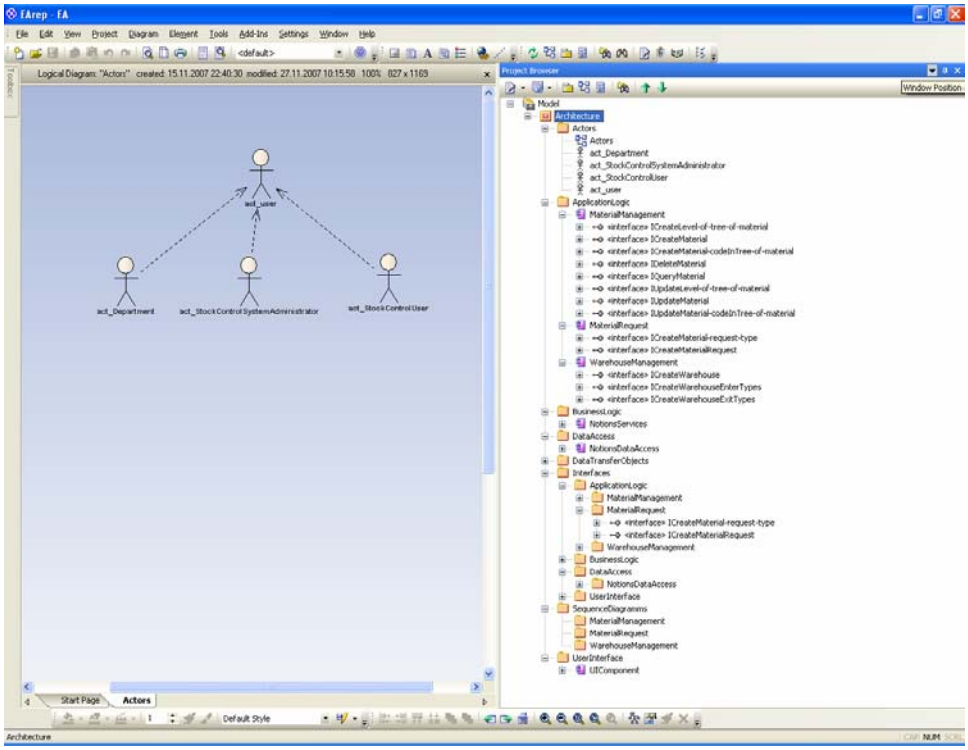
Bunun sonucunda oluşturulan RSL dosyası kullanılarak, MOLA Model Çevrim [10] üzerine bina edilen METAclipse/Eclipse temelli Mola aracı ile Şekil 9'da görülen çevrim işlemi gerçekleştirilir ve çok katmanlı Mimari Model oluşturulur.



Şekil 9. Şekil 7 ve 8 sonuçlarının Mola Tool kullanılarak, Mimari Model'e çevrilmesi

Transformasyon profili; gereksinim paketlerinde gruplanmış mantıksal ilişkili kullanım durumları, kullanım durumu senaryoları içerisindeki aksiyonların sırası ve nosyon paketlerinde gruplanmış mantıksal ilişkili nosyonlardan oluşmaktadır. Çok katmanlı Mimari Model'de paketler, aktörler, uygulama mantığı, iş mantığı, veri girişi katmanı, veri transfer nesnesi ve bileşenler oluşturulmuştur. Her bir işlevsel gereksinim paketi için bir uygulama mantığı bileşeni yaratılmıştır. Her bir kullanım durumu için de ayrıca

uygulama mantığı bileşenine göre bir arayüz oluşturulmuştur. Ortaya çıkan mimari model Şekil 10'da gösterilmektedir.



Şekil 10. Çok katmanlı ve bileşen temelli Mimari Model

5 Değerlendirme

ReDSeeDS Projesinden en önemli beklenti yeniden kullanılabilir ürünleri oluşturmak için harcanan çabayı azaltmak, böylece yeniden kullanım maliyetini düşürmek ve ürün kalitesini artırmaktır. ReDSeeDS motorunu, Cybersoft tarafından geliştirilen AURORA yazılım üretim bandı [6, 7] ile birleştirerek, AURORA'nın tekrar kullanım olanaklarını önemli ölçüde geliştirmeyi hedeflemekteyiz.

ReDSeeDS'den sağlanan ve sağlanacak faydalar iki ana başlık altında incelenebilir:

- Şirket içinde yazılım geliştirme yaşam döngüsünü gereksinim temelli hale dönüştürmek ve yeniden kullanın ile maliyetleri azaltmak: Cybersoft'un tasarladığı ve geliştirdiği Türkiye Gelir İdaresi Başkanlığı ve AVIS-Azerbaycan Vergi Nazırlığı Otomasyonu sistemleri ve diğer ülkeler için üzerinde çalıştığı fırsatlar bulunmaktadır. Bu sistemler benzer gereksinimler

içermekte olduğundan ReDSeeDS motoru kullanımından yararlanma imkanına sahiptir. ReDSeeDS yaklaşımı ile Cybersoft her bir proje için kendine özgü uyarlanmış gereksinimlerden çok, tekrar kullanılabilir, tamamlanmış, test edilmiş, gereksinim güdümlü yazılım varlıkları oluşturmayı hedeflemektedir. Uygulamanın diğer bir etki alanı büyük ölçekli Bankacılık Ürün Ailesi'dir. Bütün bu projeler AURORA uygulama platformu üzerine kurulmuştur. ReDSeeDS yaklaşımının getireceği avantajlar ile bu proje ve ürünlerde asgari %20 verimlilik artışı olabileceği gözlenmektedir.

– Bakım süreci yönetimi: Garanti süresinden sonra genellikle müşterilerin Bilgi Teknolojileri departmanları geliştirilmiş sistemin bakımından sorumludur. Bu departmanlara program hakkında özellikle yazılım geliştirme teknolojileri ve metodolojileri hakkında detaylı eğitim verilmektedir. Fakat gereksinimlerdeki sık değişiklikler ve mevzuat sebebiyle kısa zaman içinde değişiklik yapabilmek zorlu olabilmektedir. ReDSeeDS motoru ve metodoloji eğitimi ile ilgili yazılım durumlarını sağlamak, bakım süreci sırasında önemli yararlar sağlayacaktır.

6 Sonuç ve Gelecek Çalışmalar

Cybersoft tarafından Tedaş BYS Projesi Stok Kontrol Sistemi uygulaması özelinde gerçekleştirilen örnek uygulama çerçevesinde gereksinimler RSL editör ile tanımlanmış, transformasyon dilinin transformasyon kuralları ve geliştirilmiş modeller arasındaki uygulanabilirliği sınanmış, Mola Transformasyon Aracı ile mimari model oluşturulmuştur. Projenin devam eden aşamalarında oluşturulan test durumları çerçevesinde geliştirilmesi devam etmekte olan ReDSeeDS motorunun daha ileri aşama testleri gerçekleştirilecektir.

Gelinen bu noktada yapılan değerlendirme sonucu ReDSeeDS çerçevesinde önerilen yaklaşımın, yazılımın yeniden kullanımı için mevcut durumda gereksinim tanımlarını dikkate almayan çözümle sınırlı mevcut yaklaşımlara oranla önemli iyileştirmeler getirdiği görülmektedir. Genişletilen yazılım durumu tanımı çerçevesinde gereksinimler yazılım sistemlerindeki benzerlikleri ve farklılıkları tanımlamak için kullanılmaktadır. Bu yolla hem yazılım geliştirme hem de bakım ve iyileştirme projelerinde yüksek seviyede yeniden kullanımın olası hale gelebileceği pratik olarak gözlemlenmektedir. Mevcut yazılım üretim bandı ile entegre edebildiğinde; Cybersoft ReDSeeDS yaklaşımı ile yeniden kullanılabilir ürünleri oluşturmak için harcadığı çabayı azaltmayı, yeniden kullanım maliyetini düşürmeyi ve yeniden kullanım sonucunda ürün kalitesini artırmayı olası kılabilecektir.

Bundan sonraki süreçte ReDSeeDS metodolojisi ve süreç yönetim yaklaşımlarına entegrasyonu üzerinde çalışılacaktır. Özellikle yükselen eğilimler olan Servis Temelli Mimari (SOA) ve İş Süreçleri Yönetimi (BPM) kapsamındaki mimari modeller için gerekli düzenlemelerin ReDSeeDS çerçevesine yansıtılması planlanmaktadır. Cybersoft da proje bünyesindeki görevleri doğrultusunda bu eklentileri sınayacak ve kendi AURORA yazılım üretim bandına entegre edecektir.

Bilgilendirme

ReDSeeDS projesi; Infovide koordinasyonunda Varşova Teknoloji Üniversitesi, Konlenz-Landau Üniversitesi, Viyana Teknoloji Üniversitesi, Fraunhofer IESE, Litvanya Universitesi, Hamburg HITeC e.V. c/o Üniversitesi, Heriot-Watt Üniversitesi, PRO DV, Cybersoft ve Algoritmu Sistemos taraflarından yürütülmektedir.

Kaynakça

1. ReDSeeDS Proje Web Sitesi , http://www.redseeds.eu/
2. Clements, P., Nortrop, L., “Software Product Lines: Practices and Patterns”, Addison Wesley, 2001.
3. Smialek, M., “Software Development with Reusable Requirements-Based Cases”, Warsaw University of Technology, 1997.
4. Smialek, M., “Towards a Requirements Driven Software Development System.” In: Models 2006, 2006.
5. Wolter, K., Hotz, L., Krebs, T., “Towards Integration of Modeling and Reusing Software Cases”, Proc. Workshop on Software and Services Variability Management - Concepts, Models and Tools, Helsinki, Finland, 2007.
6. Altintas, N. I., Surav, M., Keskin, O., and Cetin, S., “Aurora Yazılım Üretim Bandı”, 2. Ulusal Yazılım Mühendisliği Sempozyumu, Ankara, 2005.
7. Cetin, S., “Impact of Architectural Concerns on Continual Achievement of Enterprise Applications” , 1. Ulusal Yazılım Mimarisi Konferansı, İstanbul, 2006.
8. Smialek, M., “Can Use Cases Drive Software Factories?”, 2nd International Workshop on Use Case Modeling (WUsCaM-05), MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005.
9. Altintas, N. I., Cetin, S., "Integrating a Software Product Line with Rule-Based Business Process Modeling", TEAA-2005, Trends in Enterprise Application Architecture Workshop, VLDB-2005 Conference, LNCS 3888, Trondheim - Norway, 2005.
10. MOLA Transformation Web Sitesi , http://mola.mii.lu.lv/tool_description.html

YENİ NESİL UYGULAMALAR İÇİN MİMARİ YAKLAŞIMLAR

Ortamdan-Haberdar Bir Sistem İçin Mobil Ağ Yazılım Mimarileri [1]

Halûk Gümüşkaya[1], Ahmet Volkan Gürel[2], Mustafa Veysi Nural[3]

[1,2,3] Fatih Üniversitesi, Bilgisayar Mühendisliği Bölümü, 34500, Büyükçekmece, İstanbul
[1] haluk@fatih.edu.tr, [2] avgurel@fatih.edu.tr, [3] mnural@st.fatih.edu.tr

Özet. Bu çalışmada ortamdan haberdar (context-aware) bir sistem için, sunucu ve mobil haberleşme istemci cihazları için, sistemden bağımsız, taşınabilir ağ programları ve dağıtık sistemler için yazılım mimarileri önerilmektedir. Önerilen sistemin değişik kısımları Service-Oriented Wireless Context-Aware System (SOWCAS) projesi adı altında geliştirilmektedir. SOWCAS bina içinde ve dışında çalışan, ortamdan haberdar kablolu ve çok farklı özellikleri olan kablosuz ağlar üzerinde çalışan dağıtık bir sistemdir. SOWCAS istemcilerinin sisteme bağlantısında, uygulamaların özelliklerine göre TCP/UDP soket bağlantıları, Java RMI dağıtık nesne teknolojisi ve SOAP mesajları ve Web Servisler ile servis odaklı mimari yaklaşımlar kullanılmaktadır.

1 Giriş

Günlük hayatımızla bütünleşen, cep bilgisayarları, akıllı mobil telefonlar ve kablosuz algılayıcılar (sensors) gibi, üzerinde birçok özellik içeren mobil ve kablosuz haberleşme cihazlarının yetenekleri ve kullanımı hızlı bir şekilde artmaktadır. Ayrıca, bu cihazları birbirleriyle ve daha büyük bilgisayarlarla entegre ederek yaygın bilgisayar sistemlerine bütünleştiren, IEEE 802.11abg/n, Bluetooth, ZigBee ve Wimax gibi kablosuz ağ çözümleri gittikçe yaygınlaşmaktadır.

Bu gelişmelerin neticesi, günümüzde dağıtık sistemler, çok yaygın (pervasive) bir hale gelmekte ve yaşamın birçok alanında önemli etkileri olmaktadır. Günümüzün dağıtık sistemleri, çok küçük kablosuz algılayıcılardan oluşan geçici (ad-hoc) ağlardan başlayıp, eşli (peer-to-peer) sistemler gibi katmanlı (overlay) ağlara veya çok güçlü sunuculardan oluşan sunucu ağlarına (massive web farms) kadar uzanmaktadır.

Mobil cihazlar, kablosuz erişim teknolojileri ve dağıtık sistemlerdeki gelişmeler, *yaygın bilgi işlemede (pervasive computing)* ve *ortamdan-haberdar (context-aware)* sistemlerde yeni gelişmelere neden olmaktadır. Bütün bu gelişmeler, mobil cihazlar ile bina içi ve bina dışındaki dağıtık ve kablosuz ortamlar için yeni yazılım tekniklerinin ve mobil servislerin geliştirilmesini gerektirmektedir.

[1] Bu çalışma P50050702 proje numarası ve *Frameworks for Context-Aware Pervasive Systems─FCAPSYS* adı ile Fatih Üniversitesi Bilimsel Araştırma Fonu tarafından desteklenmektedir.

2 Önceki Çalışmalar

Ortamdan Haberdar Bilgi İşleme (OHBİ) mobil bilgi işlemenin oldukça yeni bir dalı olup, kendi kullanım ortamlarını algılayabilen ve buna göre davranışlarını uyarlayabilen mobil sistemleri kapsamaktadır. OHBİ kullanıcıların çevreleriyle daha iyi etkileşim kurmasına yardımcı olur. OHBİ fikri, yaygın bilgi işleme (ubiquitous computing) konusundaki öncü çalışmalardan birinde [1] sunulan ilk kavramlardan olmuştur ve o zamandan beri de uzun araştırmalara konu olmaktadır.

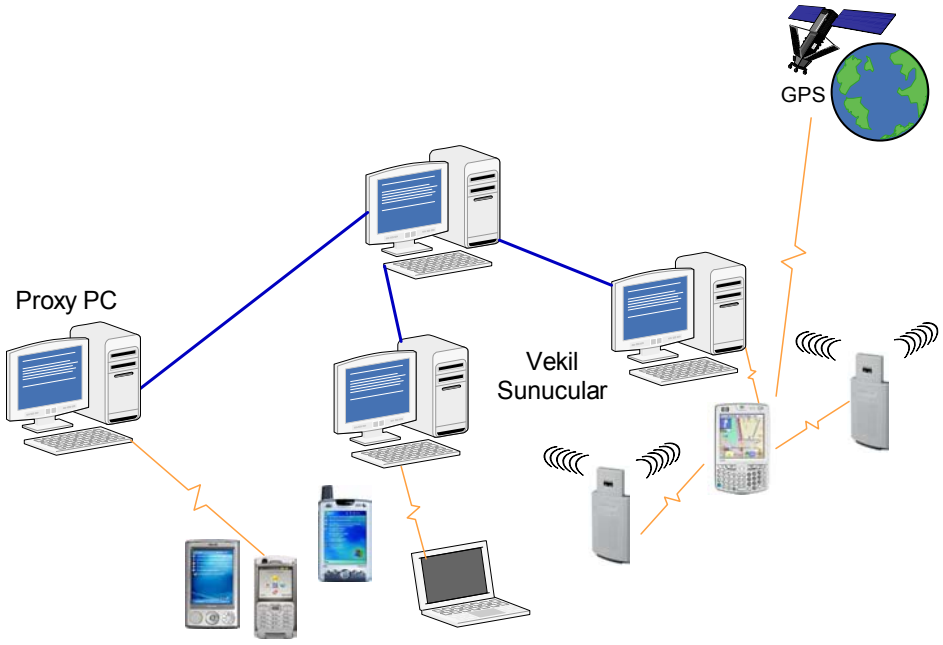
Mobil cihazlar ve dağıtık sistemlere yönelik varolan ve yeni çıkan standartların kullanıldığı *Service-Oriented Wireless Context-Aware System (SOWCAS)* [2] adında ortamdan haberdar bir sistem (framework) geliştirmekteyiz. SOWCAS'daki temel amacımız, bir üniversite kampüsündeki kablosuz erişim teknolojilerinin heterojenliğinden yararlanan, bina içinde ve dışında çalışan ortamdan haberdar bir sistem geliştirmektir.

SOWCAS'da yeni yaklaşımları ve daha önceki projelerimizde geliştirdiğimiz bazı tasarım fikirlerini de iyileştirerek kullanmaktayız. Daha önceki bir çalışmamızda [3] *desteklenebilir (supportable)*, yani anlaşılabilir, bakılabilir (maintainable), ölçeklenebilir ve taşınabilir ön (meta) yazılım mimarisi ve bazı tasarım prensipleri önerdik. Bu mimari ve tasarım prensiplerini, konum bulma sistemimiz WiPOD *(Wireless Position Detector)* [4] üzerinde uyguladık. WiPOD'da farklı algoritmalar kullanarak olası WLAN konum bulma prototiplerini inceledik ve bunların gerçek hayatta kullanılacak doğrulukta sonuçlar verip vermediklerini araştırdık. WiPOD'dan sonra *Service-Oriented Reflective Wireless Middleware (SORWiM)* [5] adında, bina içi ve bina dışı mobil uygulamalar için genel (generic) bir sistem geliştirdik. Sistem, mobil bilgi işleme için etkin ve güvenilir bilgi bulma ve dağıtma amaçlı bir dizi temel (basic) ve birleşik (composite) web servisleri sunmaktadır.

En çok araştırılan ortam bilgisi konumdur (location). Ortamdan haberdar sistemlerin büyük bir kısmı konum bulma sistemlerinin üstüne bina edilmiştir. Böylece belirli özel işlemler için ortamdan haberdar bilgi sağlanmaktadır. Kullanıcının konumunu bulma, ortamdan haberdar bir sistem için ilk ön şarttır. SOWCAS konum bulma motoru olarak WiPOD'u kullanmaktadır. WiPOD bina içinde ve dışında mobil kullanıcıların konumunu belirlemek için çok bilinen Triangulation ve K-Nearest Neighbor (KNN) algoritmalarını kullanmaktadır. SOWCAS'da kullanılan tasarım metodolojisi ve mimarisinden dolayı, Place Lab [6] gibi diğer açık kaynak kodlu konum bulma sistemleri, bizim sistemimize entegre edilebilir. WiPOD, Fatih Üniversitesi kampüsünde IEEE 802.11, Bluetooth ve GPS destekli cihazı olan kullanıcıların konumlarını belirler ve takip eder. Sistem çok sayıda WLAN erişim noktasından aldığı sinyalleri ve uydulardan gelen GPS verilerini işleyerek çalışır. SOWCAS sunucusundaki merkezi veritabanında, bina içi ve bina dışı konum bilgilerinin yanında, öğrenciler ve fakülte üyelerinin durum, faaliyet, haftalık iş takvimi ve ayrıntılı kişisel bilgileri de kaydedilmektedir.

3 SOWCAS Sistem Mimarisi

SOWCAS'da ortam bilgilerini toplamak ve işlemek için, Şekil 1'de görüldüğü gibi, merkezi bir SOWCAS Sunucusu, Vekil (Proxy) PC'ler ve mobil SOWCAS istemcilerinden oluşan dağıtık bir mimari tasarladık ve gerçekleştirdik. SOWCAS Vekil yazılımı, sınıflar, laboratuvarlar ve üniversitedeki diğer bazı özel yerler gibi, öğrenci ve fakülte öğretim üyelerinin sıkça görüldüğü yerlerde bulunan Vekil PC'ler üzerinde çalışmaktadır. WLAN 802.11 ve Bluetooth ağ kartlarına sahip bu PC'ler ile binalardaki kablosuz erişim noktaları, üniversitedeki öğrenci ve fakülte üyelerinin yerlerini belirlemek ve takip etmek için kullanılmaktadır.



Şekil 1. SOWCAS sistem mimarisi

Bina içi ve bina dışı konum ortam bilgisinin yanında, öğrenciler ve öğretim üyeleriyle ilgili merkezi veritabanında, Tablo 1'de görülen, durum (status), faaliyet (activity), haftalık programlar ve detaylı kişisel bilgiler tutulmaktadır.

Tablo 1. Fakülte üyeleri ve öğrenciler için tipik ortam (context) bilgilerinden bazıları

Durum	Faaliyet	Diğer Bilgiler
Kapalı / Açık (Offline / Online)	Üniversite Kampüsüne Gelmekte	Haftalık Programlar
Uygun / Uygun Değil (Available / Not Available)	Üniversite Kampüsünden Ayrılmakta	Kişisel Bilgiler
Uzakta (Away)	Sınıfta / Lab'da	
Meşgul (Busy)	Toplantıda	
Görünmez (Invisible)	Kafeteryada Yemekte, Dinleniyor	
	Ofiste/Sınıfta/Kütüphanede Çalışıyor	
	Otomat/ATM Makinesi Kullanıyor	

4 Mobil Cihazlara Yönelik Yazılım Mimarileri

4.1 Küçük Mobil Haberleşme Cihazlarının Genel Özellikleri

PDA, telefon ve sensörler gibi mobil cihazların en yaygın özellikleri arasında, sınırlı kaynaklar (mikroişlemci, hafıza, giriş/çıkış yetenekleri gibi), heterojenlik ve yüksek düzeyde dinamizm gelmektedir. Kaynakların sınırlılık durumları, cihazdan cihaza değişkenlik gösterse de, mobil cihazlar genellikle masaüstü bilgisayarlar kadar güçlü işlemcilere, büyük hafızalara, yüksek hızlı G/Ç arabirimlerine ve ağ yeteneklerine sahip değildir. Tipik günümüz PDA CPU hızları 300-500 MHz arasındadır. Yaygın olarak 64-128 MB ana hafızaya, 1-4 GB harici flash hafızaya, USB 1.1 desteğine sahiptirler. Mobil telefonlar GSM/UMTS/GPRS/EDGE gibi hücresel mobil haberleşme teknolojilerine sahiptirler. Günümüz mobil haberleşme cihazlarında yaygın olarak Bluetooth 1.2 ve 802.11b (11 Mbit, özellikle cep bilgisayarlarında) desteği ile çok azında da 802.11g (54 Mbit) ve GPS desteği bulunmaktadır. Farklı donanım/yazılım platformları ve işletim sistemlerinde, byte sıralaması, standart tiplerin byte uzunluğu ve haberleşme protokolleri gibi bazı parametreler değişiklik göstermektedir.

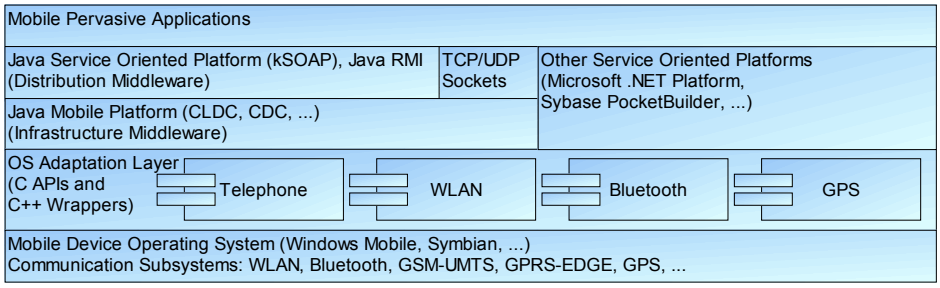
Tüm yukarıda sayılan özellikler mobil bilgi işleme için gerekli *kablosuz ortakatmanın (wireless middleware)* tasarımını etkilemektedir. RMI, CORBA gibi geleneksel ortakatman platformları ve SOAP gibi yeni teknolojiler bazı mobil cihazlar ve kablosuz uygulamalar için uygun olmayabilir. Çünkü, öncelikle bu teknolojiler, günümüz mobil cihazlarının birçoğu için fazla büyüktür ve esnek değildir. Bir kablosuz ortakatmanın kaynakları kısıtlı el cihazlarında çalışabilmesi için hafif olması gerekmektedir. Geleneksel ortakatman platformları, kaynakları çok değişiklik gösteren kablosuz ağlarda, sabit bağlanırlık (static connectivity), güvenilir (reliable) kanallar ve yüksek bant genişliği gibi özelliklere ihtiyaç duyar. Mobil cihazlar imkan buldukça ve kısa sürelerle ağ bağlantısı kurduğundan, kablosuz ara katman eş zamanlı olmayan (asenkron) haberleşme şekillerini desteklemelidir. Ayrıca en iyi hizmet kalitesi ve en uygun kaynak kullanımı sağlamak için, ortakatman yazılımı haberdar olma prensibine (awareness principle) göre tasarlanmalıdır ki, uygulamalarının kendini ve ortakatman davranışlarını yürütme ortamındaki değişikliklere göre uyarlamasına izin verebilsin.

Uygulamalar ve ortakatman, konum, bağlanabilirlik, bant genişliği ve batarya gücündeki değişikliklere göre kendini uyarlamak zorunda olduğundan, ağ topolojilerini ve diğer uygulama ayrıntılarını, dağıtık uygulamalardan gizlemek (location transparency) oldukça zordur ve klasik kablolu ağlarda çalışan dağıtık sistemlerdeki gibi istenmemektedir.

4.2 SOWCAS İstemci Mimarisi

SOWCAS istemci ve sunucu tasarımlarında ortakatman yaklaşımını kullandık [7]. SOWCAS ortakatmanı farklı alt katmanlara bölünmektedir: Alttan yukarı doğru, cihaz altyapı (host infrastructure) ortakatmanı (Java Virtual Machine (JVM) gibi), dağıtım (distribution) ortakatmanı (RMI ve SOAP gibi), genel ortakatman servisleri ve alana özel (domain-specific) ortakatman servisleri [2].

Yeni gelişmiş cep bilgisayarları ve akıllı telefonlar GSM/UMTS, GPRS/EDGE, WLAN, Bluetooth ve GPS gibi heterojen kablosuz teknolojilere sahip. Bu kablosuz teknolojilere programlama ile erişmek için genellikle C/C++ API'leri (Application Programming Interfaces) gibi sisteme bağımlı üçüncü parti ürünler ve yazılım geliştirme ortamları kullanılmaktadır. Bu haberleşme katmanı, Şekil 2'deki SOWCAS istemci mimarisinde İşletim Sistemi Adaptasyon Katmanı (Operating System Adaptation Layer) olarak gösterilmiştir.



Şekil 2. SOWCAS istemci mimarisi

SOWCAS'ın istemci ve sunucu kısımlarında taşınabilirlik ve yazılım geliştirmeyle ilgili nedenlerden dolayı mümkün olduğunca Java kullanmaktayız. Bazı mobil cihazlar için Java kullanımı mümkün ya da uygun olmadığında, istemci uygulamalarını geliştirmek için Microsoft .NET ve Sybase PocketBuilder gibi diğer yazılım geliştirme platformlarını kullanıyoruz.

Farklı SOWCAS istemci uygulamaları için, kısa mesafeli Bluetooth ve IEEE 802.11 gibi kablosuz ağ erişimlerinde, referans haberleşme protokollerinin ve mimarilerinin seçilmesi amacıyla, temel TCP/UDP soketleri, Java RMI ve SOAP tabanlı küçük istemci test programları geliştirildi. Bu test programları ve bu farklı yazılım teknolojilerinin ayrıntılı performans analizleri diğer çalışmamızda sunulmaktadır [8].

Bu yapmış olduğumuz analizlerin sonuçlarına göre, farklı uygulamalarda kullanılacak sunucu ve istemci ağ bağlantı teknolojilerine karar verilmiştir.

SOWCAS sisteminde mobil istemcilerdeki önemli bir problem, hareketli bir kullanıcının konum ve durumuyla ilgili ortam (context) bilgilerinin sürekli sağlanacak bir kablosuz ağ bağlantısı ile sunucuya gönderilmesidir. IEEE 802.11 stardardı [9], [10] erişim noktaları (access point) arasında veri bağı katmanında (data link layer) bağlantı aktarmayı (handoff) tanımlamasına rağmen, incelemiş olduğumuz birçok küçük haberleşme cihazlarında ve taşınır bilgisayarlarda, farklı firmaların erişim noktaları arasında otomatik olarak erişim noktası değiştirme özelliğine rastlamadık. Yani kullanıcı mobil cihazı bir erişim noktasına bağlandıktan sonra hareket ederse ve başka yeni bir erişim noktasına yaklaşırsa, sinyal gücü yüksek olan bu yeni erişim noktasına kullanıcı müdahalesi olmadan cihaz otomatik bağlanmıyor. Mobil telefonlar, kullanıcı hareket ederken, kullanıcıya hissettirmeden ve telefon görüşmesini kesmeden baz istasyonlarını değiştirdiği gibi, bir mobil bilgisayarın da bir IEEE 802.11 ağı içinde kullanıcıya saydam, hissettirmeden ve ağ bağlantısını kesmeden bağlantı noktasını değiştirmesi istenmektedir. Bu özelliğin elimizde mevcut olan cihazlarda sağlanmadığını görünce, bunu gerçekleştirmek için bir proje başlattık [11] ve projemiz başarılı bir şekilde bitmiştir. Bu projemiz Linux işletim sisteminde çalışan ve Wireless Tools for Linux [12] yazılım paketlerini kullanarak bağlantı aktarmayı (handoff) sağlayan bir yazılım katmanını gerçekleştirmektedir ve SOWCAS mobil istemci yazılım mimarisindeki alt katmanlara eklenmesi hedeflenmektedir.

5 SOWCAS Sunucuları

5.1 Kablosuz Ortakatman Sunucusu

Aksettirici ortakatman (reflective middleware), yaygın bilgi işleme (ubiquitous computing) için kapsamlı bir çözüm sunmaktadır [13]. Aksettirici ortakatman sistem cevapları; mobil bağlantılar, güç düzeyleri, ağ bant genişliği, gecikme süresi (latency)/paketler arası zaman farkları (jitter) ve güvenilirlik (reliability) ihtiyaçları gibi değişken ortamlara göre optimize edilir. Aksettirme (reflection), bir programın yapı ve davranışını gözlemleyebilme ve imkan dahilinde değiştirebilme yeteneğidir.

Bir aksettirici ortakatman sunucusu, farklı ağ uygulamalarının gerektirdiği, değişik ağ bant genişliği, veri güvenilirliği (reliability) ve zamana hassasiyet (time sensitivity) isteklerine göre, farklı ağ bağlantı seçenekleri (TCP/UDP, RMI ve SOAP gibi) sunması gerekir. Ayrıca istemcilerin durumlarına, örneğin donanım kaynaklarına (işlemci ve hafıza gibi), pil güçlerine ve ağ bağlantı kalitesine göre farklı hizmetleri sunması gerekir.

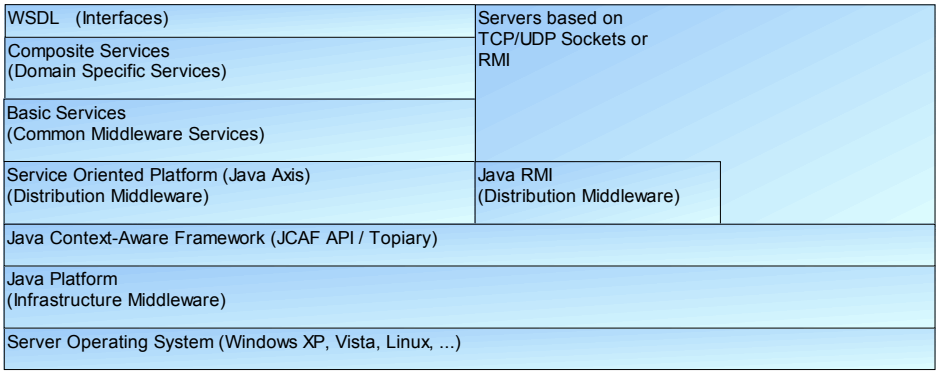
Yukarıda özet olarak sunulan SOWCAS mimarisi geliştirilirken, küçük mobil cihazlar ile yapılan kısa mesafeli kablosuz bağlantılardaki, temel TCP/UDP soket bağlantıları, RMI dağıtık nesne teknolojisi ve servis odaklı yaklaşım kullanılarak istemci/sunucu mimarileri incelendi. Geliştirilen test programları ile bir istemci ile sunucu arasındaki değişik kablosuz ağ bağlantılarının paket ölçümleri ve zaman analizleri detaylı olarak yapıldı [8].

Servis Odaklı Bilgi İşleme (SOBİ), servisleri kullanan yazılım uygulamalarının geliştirilmesi için mimari model olan Servis Odaklı Mimari (SOM) tabanlı bir dağıtık bilgi işleme yaklaşımıdır. SOBİ ve SOM tamamen yeni kavramlar değildir, CORBA ve RMI gibi diğer dağıtık bilgi işleme teknolojileri de benzer kavramlar kullanmaktadır. SOBİ ve SOM var olan kavramların ve XML, SOAP gibi platformdan bağımsız dağıtık sistemlerin gerçekleştirilmesinde kullanılan yeni teknolojilerin devamı niteliğindedir.

SOM mobil servisler için ideal bir yaklaşım olarak görülmektedir [2], ancak şu anda kurumsal ve ticari servisler üzerinde odaklanmıştır. Ayrıca, SOM ile ilgili araştırmaların büyük bölümü kablolu ağlara yönelik mimariler ve uygulamalar üzerinde odaklanmıştır. SOM tabanlı kablosuz ortakatman ile ele alınması gereken birçok problem bulunmaktadır. Kablosuz ortakatman, servis odaklı uygulamaların hazırlanması ve yönetilmesinde çok önemli bir rol oynayacaktır. Bu araştırmanın ana hedeflerinden birisi, mobil servislerin gerçekleştirilmesinde SOM yaklaşımından nasıl yararlanılabileceğini incelemektir.

5.2 SOWCAS Sunucuları

Şekil 3'de SOWCAS sunucularının genel mimarisi basitleştirilmiş olarak verilmektedir. SOWCAS sunucusu istemcileri ile uygulamaların yapısına göre Web Servisler, RMI ve TCP/UDP soketleri ile haberleşmektedir. Bu sağlanan haberleşme yapıları mimarinin en üst katmanında görülmektedir.



Şekil 3. SOWCAS sunucu mimarileri

Sistemde daha önceden Event, Location, Messaging, Redirection, Communication, Multimedia gibi temel servisler yazıldı [5]. Birleşik (composite) servisler bu temel servislerin kullanımıyla oluşturuldu. Bu temel servisler ve birleşik servisler Şekil 3'te sol üstte, paylaşılan ortakatman servisleri (Common Middleware Services) ve ortama özel servisler (Domain Specific Services) olarak gözükmektedir.

Daha önce bahsetmiş olduğumuz SOWCAS konum motoru (positioning engine) WiPoD sunmuş olduğu konum servisi (Location Service) ile SOWCAS sunucusunda bulunmaktadır. Bina içinde ve dışında geniş alanlar için doğru ve gereken hassasiyette konum bulma problemlerine çalışırken WiPoD'u iyileştirmek yerine, daha üst seviyeli problemlerle uğraşmak için, bu konum motorumuzun yerine değişik alternatifler üzerinde çalışmaktayız. Yakın bir zamanda programlanabilir ticari bir konum bulma sistemi üzerine sistemimizi taşımayı planlamaktayız.

İlk SOWCAS prototipinde, sistemdeki ortam (context) bilgilerinin modellenmesinde ve işlenmesinde, Java RMI tabanlı ortamdan haberdar bir sistem olan JCAF'ı (Java Context-Awareness Framework) kullandık [14]. RMI özelliği JCAF istemcisinin Java diline bağımlı olmasını gerektirmektedir. Şekil 3'de görüldüğü gibi, mimarimizde JCAF'ın üzerinde web servis katmanı şeklinde bir katman daha bulunduğundan, sistemimiz platformdan bağımsızdır. JCAF bize birçok sınırlama getirdiğinden, Topiary projesindeki [15] bazı fikirleri ve yazılım bileşenlerini de kullanarak kendi ortamdan haberdar sistemimizi geliştirmeye başladık.

6 SOWCAS Uygulamaları

Bu bölümde SOWCAS mimarisi üzerinde geliştirilen ve şu an devam eden veya yeni biten bazı projelerimiz kısaca tanıtılacaktır.

6.1 Web Servisleri Kullanan Uygulamalar

Web servis kullanan uygulamalar en üst düzeyde istemci heterojenliğini sağlayan, yani çok farklı mobil donanım ve işletim sisteminde çalışan uygulamalara yönelik, çok hız gerektirmeyen uygulamalar için kullanılmaktadır. Şekil 4 (a) ve Şekil 4 (b)'de SOWCAS projesinden önce geliştirilen [5] iki mobil web servis uygulaması görülmektedir. Şekil 4 (a)'daki uygulama, farklı kablosuz ağ bağlantılarında geçiş yapmak için (örneğin, GPRS bağlantısından bir 802.11 ağına geçmek gibi) kullanılan bir web servistir. Şekil 4 (b)'deki uygulama, bir kullanıcının (örneğin bir öğrenci) üniversite alanına girdiğinde bundan haberdar olmak isteyen diğer bir kullanıcı için (örneğin bu öğrencinin bir arkadaşı veya üniversite hocası) geliştirilen konum tabanlı haber verme servisidir. Bu tür uygulamalar için web servislerin çok uygun olduğunu projemizde gördük.

Şekil 4. SOWCAS uygulamalarından bazı örnekler

6.2 RMI Uygulamaları

RMI Java'da hızlı ağ programları geliştirmek için çok uygun bir teknolojidir. Java tabanlı istemci makinelerine hizmet eden, bazı RMI sunucu programları geliştirildi. Üniversitedeki öğrencilerin merkezi SOWCAS sunucusunu kullanarak kampüs içindeki arkadaşlarını bulmalarını ve birbirleri ile eşli (Peer-to-Peer) bağlantılar kurarak mesaj göndermelerini ve dosyalarını paylaşmalarını sağlayacak bir proje tamamlanmıştır [16].

6.3 TCP Soket Bağlantılı Uygulamalar

Yeni bitirilmiş olan diğer bir projemizde SOWCAS sistemine bir GPS servisi eklenmiştir [17]. Bu yazılım ile GPS desteğine sahip mobil bir cep bilgisayarı veya bir cep telefonu olan öğrenci veya kampüs ziyaretçisi, kampüse girişteki güvenlik noktasındaki sunucudan cihazına indireceği haritada, kampüs içinde gitmek istediği yeri kampüs haritası üzerinde bulabilecek, Şekil 4 (c)'de görüldüğü gibi, programın sağlayacağı harita ve kısa yol desteği ile gideceği yere hızlı bir şekilde ulaşabilecektir. Hareket halindeki kullanıcının GPS konum bilgisi, cihazın aynı zamanda sağlayacağı WLAN ağ bağlantısı ile, sürekli olarak bir TCP soket bağlantısı üzerinden SOWCAS veritabanına yazılmaktadır. Bu sayede sistemdeki diğer kullanıcıların bu hareketli kullanıcının konum bilgisine erişmesi mümkün olmaktadır. Bu ve benzeri uygulamalar gibi, hızlı ve basit bilgilerin sunucuya gönderilmesini gerektiren uygulamalar, TCP soket bağlantısı ile SOWCAS sunucuna bağlanmaktadır.

6.4 UDP Soket Tabanlı Uygulamalar

UDP bağlantısız ve güvenilir olmayan bir aktarım katmanı protokolüdür ve genellikle paket kayıplarına toleranslı ve hıza duyarlı, ses ve görüntü gibi, akan çoklu ortam (streaming multimedia) uygulamalarında kullanılmaktadır. Çalışmamızda RTP (Real Time Protocol) ve RTSP (Real Time Streaming Protocol) gibi uygulama protokollerini kullanarak istemcilerine ses ve görüntü verileri dağıtan serviste UDP soketleri ve datagram'lar kullanılacaktır.

7 Sonuç

Bu bildiride 2005'ten bu yana, WiPOD [4] ve SORWiM [5] ile başlayan ve diğer küçük projelerle uzun bir süredir devam eden, en son SOWCAS adı altında daha önceki çalışmaları toparlamayı hedefleyen projemizin mimarisi genel hatlarıyla tanıtıldı. Haziran 2007 tarihinden bu yana da Fatih Üniversitesince FCAPSYS adı ile projelerimiz desteklenmektedir ve bu destek Haziran 2009'a kadar sürecektir.

Projemizle ilgili şu an tamamlanmak üzere olan bir yüksek lisans tezi [18] ve üç tane yeni biten son sınıf bitirme tezi ve devam eden diğer çalışmalarımız bulunmaktadır. Daha önce bahsedildiği gibi, sistem mimarisini bir ticari konum bulma motoru üzerine bina etme üzerine çalışmaktayız. Bu sayede üniversiteye yönelik ortamdan-haberdar uygulamaları geliştirmemiz hızlanacaktır. Ayrıca mobil cep bilgisayarları ve cep telefonlarıyla beraber, bir kredi kartı büyüklüğünde olan ve pili üç dört yıl gibi uzun süre dayanan, IEEE 802.11 tabanlı aktif RFID tag'larını da SOWCAS istemcileri olarak sisteme dahil etmeyi planlıyoruz. RFID tag'larının ilk kullanım alanı olarak, 2008 yılında üniversitemizce öğrencilere verilecek 3000'den fazla taşınabilir bilgisayara RFID tag'larının yerleştirilip bunlar ile bilgisayarların takip edilmesi düşünülmektedir. Diğer yeni bir proje fikri, öğretim elemanlarının önemli bir zamanını alan, ders öğrenci yoklamalarının alınmasının bu RFID tag'ları ile otomatik veya yarı otomatik bir şekilde yapılmasıdır. Benzeri yeni projelerimizle, önümüzdeki yıl öğrencilerimiz ve öğretim elemanlarımız tarafından daha yaygın kullanılabilir ve faydalı uygulamalar geliştirebileceğimizi umuyoruz.

Kaynakça

1. Weiser, M., "The Computer for the 21st Century" Scientific American, Cilt 265, Sayı 3, Eylül 1991, s. 66-75.
2. Gümüşkaya, H., V. Nural, M., "Service-Oriented Context-Awareness and Context-Aware Services", Advances in Computer and Information Sciences and Engineering, Springer, Ağustos, 2008.
3. Gümüşkaya, H., "An Architecture Design Process Using a Supportable Meta-Architecture and Roundtrip Engineering", Lecture Notes in Computer Science (LNCS), Springer-Verlag, Cilt 4243, 2006, s. 324-333.
4. Gümüşkaya, H., Hakkoymaz, H., "WiPoD Wireless Positioning System Based on 802.11 WLAN Infrastructure", Enformatika, Cilt 9, Kasım 2005, s. 126-130.

5. Yurday, B., Gümüşkaya, H., "A Service Oriented Reflective Wireless Middleware", Lecture Notes in Computer Science (LNCS), Springer-Verlag, Cilt 4294, 2006, s. 545-556.

6. Place Lab Projesi: http://www.placelab.org.

7. Schmidt, D., "Middleware for Real-Time and Embedded Systems", Communications of the ACM, Haziran 2002, Cilt 45, Sayı 6.

8. Gümüşkaya, H. Gürel, A. V., Nural, M. V., "Küçük Mobil Cihazlarda Kablosuz Ağlar Üzerinde SOAP, RMI ve TCP Performans Analizi", 2. Ulusal Yazılım Mimarisi Konferansı (UYMK'08), 11-12 Eylül 2008, Ege Üniversitesi, İzmir.

9. IEEE. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Standard 802.11, 1999.

10. Mishra, A., Shin, M., Arbaugh, W., "An Empirical Analysis of the IEEE 802.11 MAC Layer Handoff Process", ACM Computer Communications Review, 2003, Cilt 33(2), s. 93–102.

11. Nural, M. V., Doğan, S., *Mobile Context Handoff in IEEE 802.11 Infrastructure*, Son Sınıf Bitirme Tezi, Fatih Universitesi, İstanbul, Kasım 2007 - May 2008.

12. Linux Wireless Tools: http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html.

13. Roman, M., Kon, F., Campbell, R., "Reflective Middleware: From your Desk to your Hand," IEEE Distributed Systems Online, 2001, Cilt 2, Sayı 5.

14. Bardram, J. E., "The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications", The 3rd International Conference on Pervasive Computing, LNCS, Springer Verlag, 2005.

15. Topiary Projesi: http://guir.berkeley.edu/projects/topiary/.

16. Karzaoğlu, M., Özel, G., *Mobile Ad Hoc Peer-to-Peer Context-Sharing System: P2P Messaging and File Sharing Services*, Son Sınıf Bitirme Tezi, Fatih Universitesi, İstanbul, Kasım 2007 - May 2008.

17. Parmaksız, M. F. Albayrak, T., *Context-Aware Services for Mobile Applications for Fatih University Campus: Campus GPS Service*, Son Sınıf Bitirme Tezi, Fatih Universitesi, İstanbul, Kasım 2007 - May 2008.

18. Gürel, A. V., *Service-Oriented Wireless Context-Aware System: System Architecture*, Master Tezi, Fatih Universitesi, İstanbul, Temmuz 2008.

Anlamsal Veb Servisleri Ortamında Bir Aracı Etmen Tasarımı ve Gerçekleştirimi

Özgür Gümüş[1], Önder Gürcan[2], Oğuz Dikenelli[3]

[1,2,3] Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, 35100, Bornova, İzmir
[1] ozgur.gumus@ege.edu.tr, [2] onder.gurcan@ege.edu.tr, [3] oguz.dikenelli@ege.edu.tr

Özet. Hem arabuluculuk ve koordinasyon özelliklerine sahip olmaları hem de geniş bir alanda uygulanabilir olmaları nedeniyle, aracılar anlamsal veb servisleri ortamının da doğal bir bileşeni olmaya adaydırlar. Bu çalışmada, SWSA soyut mimari çerçevesine dayalı olarak, anlamsal veb servisleri ortamında bir aracı etmen tasarlanmıştır. Bu tasarımın ana katkısı, SWSA çerçevesine uygun, esnek ve yeniden kullanılabilir bir aracı etmen mimarisi sunulmuş olmasıdır. SWSA çerçevesinin üç aşaması, o sürecin içsel gereksinimlerini karşılamak için farklı etkinlikler içermektedir ve her etkinlik, farklı şekillerde gerçekleştirilebilmektedir. Bu bağlamda, bu çalışma kapsamında tasarlanan aracı etmen, plan seviyesinde ve modül seviyesinde yeniden kullanım sağlayarak etkinliklerin farklı gerçekleştirimlerinin sisteme kolaylıkla eklenebildiği esnek bir mimari sunmaktadır.

1 Giriş

Anlamsal veb servislerinin, işlevsellikleri, erişim ve işletim yöntemleri ontolojiler kullanılarak tanımlanır. Ontolojilerin kullanımı, servislerin keşfi, işletimi v.b. yetenekleri sağlayan bir ortam gerektirir. Böyle bir anlamsal veb servisleri ortamında, aracıların kritik ve yararlı bir mimari eleman oldukları kabul edilmektedir [1, 2, 3].

Aracılar, iki ya da daha fazla taraf arasındaki etkileşimin kolaylaştırılmasına ihtiyaç duyulduğunda koordinasyon ve arabuluculuk mekanizmaları sağlarlar. Örneğin, iki taraf iletişimde bulunmak istiyor ancak ortak bir dili paylaşmıyorlarsa, aracılar çeviri servisleri sağlayabilirler; birbirlerine güvenmeyen iki taraf arasında güvenilir bir ortam oluşturabilirler. Dahası, taraflar arasındaki işlemlerin yürütülmesine arabuluculuk yaparak tarafları anonim hale getirebilirler. Ayrıca, aracılar özerk etmenler arasında keşif ve senkronizasyon mekanizmaları sağlayan ana elemanlardan birisidir [1, 4]. Hem arabuluculuk ve koordinasyon özelliklerine sahip olmaları hem de geniş bir alanda uygulanabilir olmaları nedeniyle, aracılar anlamsal veb servisleri ortamının da doğal bir bileşeni olmaya adaydırlar.

Bu çalışmada amaçlanan, etmenler alanında yapılan aracılık çalışmalarının ışığında ve SWSA (Semantic Web Services Initiative Architectural Committee) soyut mimari çerçevesine dayalı olarak, anlamsal veb servisleri ortamında bir aracı etmen tasarlamak ve gerçekleştirmektir. Bu tasarımın ana odak noktası, SWSA soyut mimari çerçevesine uygun, esnek ve yeniden kullanılabilir bir aracı etmen mimarisi sunmaktır. SWSA soyut mimari çerçevesi [5], servislerin kullanım süreci için, üç ardışık aşama tanımlamaktadır: keşif, uzlaşma ve yürütme. Her aşama, o sürecin içsel gereksinimlerini karşılamak için

farklı etkinlikler içermektedir. Burada önemli olan nokta, her etkinliğin, uygulama gereksinimlerine bağlı olarak farklı şekillerde gerçekleştirilebilmesidir. Örneğin, keşif aşamasındaki servis seçme etkinliği, servislerin anlamsal yakınlıklarına göre [6, 7], servislerin kalite metriklerine (QoS) göre [8, 9], kullanıcıların genel değerlendirmelerine (reputation-based) göre [10] ya da kullanıcıların deneyimlerine (experience-based) göre [11] servis seçimi gibi farklı şekillerde gerçekleştirilebilir. Benzer şekilde, uzlaşma aşamasında çeşitli pazarlık yöntemleri uygulanabilir. Bu bağlamda, bu çalışmada tasarlanan aracı etmen, etkinliklerin farklı gerçekleştirimlerinin sisteme kolaylıkla eklenebildiği esnek bir mimari sunmaktadır.

Bildirinin bundan sonraki bölümleri şu şekildedir: Bölüm 2'de, bu çalışmanın alt yapısını oluşturan konular açıklanmış ve literatürde yapılan ilgili çalışmalar verilmiştir. Bölüm 3'te, anlamsal servis aracılığı için bir çok-etmenli sistem platformu tanıtılmıştır. Böyle bir platformda yer alacak aracı etmenin tasarımı ve gerçekleştirimi Bölüm 4'te verilmiştir. Bölüm 5'te ise yapılan katkılar özetlenmiş, aracı etmenin mimari tasarım açısından değerlendirmesi yapılmış ve ileriye yönelik planlanan çalışmalar verilmiştir.

2 Alt Yapı ve İlgili Çalışmalar

Veb servislerinin anlamsal veb ortamında çalışması için anlamsal veb servisleri alanında bazı standartlaşma çabaları vardır. Bunlardan en ilgi çekicileri OWL-S[1] ve WSMO[2]'dur. OWL-S, veb servislerini betimlemek için bir ontoloji sistemidir. OWL-S, servislerin yeteneklerini (yapabildiklerini) ilan etmek için bir profil (*profile*) ontolojisi, servislerin işlevselliğini ve birleştirimini tanımlamak için bir süreç (*process*) ontolojisi ve servislere nasıl erişileceğinin detaylarını vermek için bir zemin (*grounding*) ontolojisi içerir. Bu ontolojiler, anlamsal yeteneğe sahip yapılar tarafından servis arama, bulma ve dinamik çağırma aşamalarında çıkarsama amaçlı olarak kullanılmaktadırlar. OWL-S, anlamsal veb servisleri kavramı için olan ilk çabadır ancak tam bir sistem değildir, bazı elemanlarını anlamı açıkça tanımlanmamıştır. Dolayısıyla, birçok araştırma grubu tarafından OWL-S ontolojisinin kullanıldığı eşleştiriciler (matchmakers), planlayıcılar ve aracılar (broker) gerçekleştirilmeye çalışılmıştır. Diğer taraftan, WSMO üst modeli dört temel eleman tanımlamaktadır: ontolojiler, hedefler, veb servisleri ve arabulucular. WSMO daha bütünleşik bir çerçevedir ancak OWL ve SWRL gibi W3C standartları tabanlı değildir. Ayrıca, daha çok dağıtık ve heterojen servis ortamında bir iş akış sistemine benzemektedir.

Bu arada, SWSA tarafından, anlamsal veb servisleri teknolojilerine altyapı oluşturması amacıyla birtakım mimari ve protokol soyutlamaları içeren bir anlamsal veb servisleri mimarisi tanımlanmıştır [5]. Bu mimari çerçeve, W3C veb servisleri mimarisi çalışma grubunun, Veb Servisleri İçin Mimari önerisi[3] üzerine inşa edilmiştir ve anlamsal servis etmenlerinin tüm gereksinimlerini karşılamaya çalışmaktadır: dinamik servis keşfi, servisle uzlaşma, servis sürecini yürütme ve ayrıca yönetim, destek ve servis kalitesi hizmetleri. Bu mimari çok etmenli sistem altyapısı temellidir. Çünkü belirtilen

[1] Semantic Markup for Web Services, http://www.daml.org/services/owl-s/
[2] Web Service Modelling Ontology, http://www.wsmo.org/
[3] W3C Web Services Architecture Working Group, Web Services Architecture Recommendation, 11 Ocak 2004, http://www.w3.org/TR/ws-arch/

gereksinimler, hedef yönelimli yazılım etmenleri kullanılarak ve tanımlanmış protokollere dayalı asenkron etkileşimler ile yerine getirilebilir.

SWSA mimari çerçevesi, bir anlamsal veb servisinin bulunması ve onunla etkileşimde bulunulmasını içeren tüm süreci üç ardışık aşamada tarif etmektedir. Aday servislerin keşfi olarak adlandırılan ilk aşama, bir istemci etmenin içsel hedeflerinin bazılarını potansiyel olarak yerine getirebilecek uygun servislerin aranmasıdır. Servisle uzlaşma adı verilen ikinci aşama, aday veb servislerinin yürütülmesi için kısıtlamaların yorumlanması ve bir anlaşmaya varıncaya kadar aday servislerle müzakere edilmesi sürecini kapsar. Bundan sonraki aşama, servisin yayınlanmış protokollerinin takip edilerek istemci ve servisin karşılıklı olarak üzerinde anlaştıkları hedefe ulaşılması için servisin yürütülme aşamasıdır. Bu aşamada, istemci servisin işletilmesi için gerekli girdileri sağlar ve servis işletiminin başarılması ya da başarılmaması durumunda ne yapacağını bilir. Ayrıca SWSA çerçevesi, her aşamanın aktörlerini, işlevsel gereksinimlerini ve bu gereksinimlerin yerine getirilmesi için gerekli olan mimari elemanları soyut protokoller açısından belirlemektedir.

Anlamsal veb servisleri ortamı için bazı aracı uygulamaları vardır: IRS-III çerçevesi [3] ve OWL-S Veb Servisleri İçin Bir Aracı [2]. IRS-III, SESA mimarisine [12] dayalı bir çerçevedir. SESA mimarisinde, Anlamsal Işletim Ortamı (Semantic Execution Environment) olarak adlandırılan ara yazılım (middleware) katmanı, mimarinin çekirdeğini oluşturmaktadır. Bu katman, mimaride belirtilen kavramsal işlevsellikleri tanımlamaktadır. Buradaki her bir işlevsellik, ara yazılım servisleri denilen bir takım servisler aracılığıyla gerçekleştirilmektedir. IRS-III, SESA mimarisinin örnek bir gerçekleştirimidir. Burada, bir servis istemci ile sağlayıcılar arasında arabuluculuk yapılarak, anlamsal veb servislerinin kullanıldığı uygulamalar oluşturulması için anlamsal aracı temelli bir yaklaşım izlenmektedir. OWL-S Veb Servisleri İçin Bir Aracı isimli çalışmada ise, bir aracı mimarisi ve OWL-S'e dayalı bir gerçekleştirim ortaya konulmaktadır. Bu mimari, gerek duyulan işlevsellikleri ve OWL-S tabanlı servis kullanım sürecini yönetmek için gereken protokolleri tanımlamaktadır.

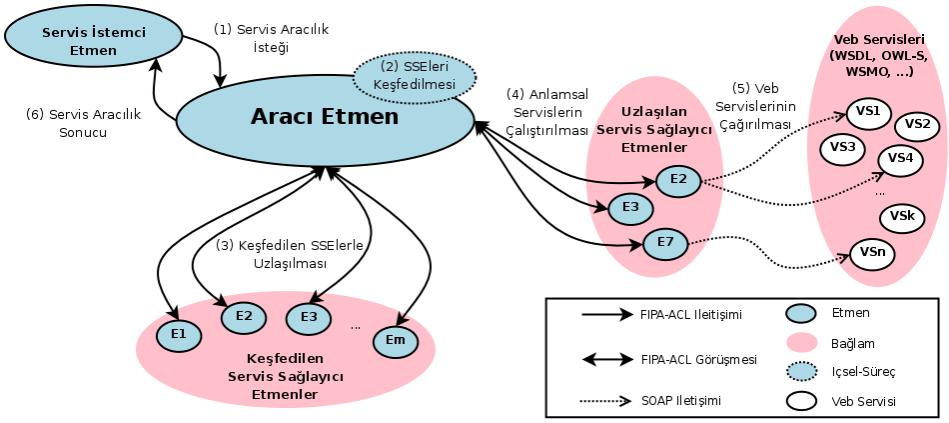
Sözü edilen çalışmalardan farklı olarak, bu çalışmada, temel mimari olarak SWSA çerçevesi alınmıştır. Çünkü SWSA çerçevesi, anlamsal servis ortamının tüm yönlerini (müzakere, anlaşma, güven v.b.) kavramsal seviyede kapsamaktadır. Anlamsal servis ortamının işlevsellikleri, SWSA'da olduğu gibi, geniş kapsamlı bir şekilde ele alındığında, böyle bir ortamı gerçekleştirmek için etmen tabanlı yaklaşımların tek çıkar yol olduğu görülmektedir. Bu çalışmada bahsedilen aracı, SWSA aşamalarını destekleyen bir etmen olarak geliştirilmiştir ve bu aşamaların etkinliklerini gerçekleştirmek için kolay ekleme yapılabilmesini sağlayan bir altyapıya sahiptir.

3 Anlamsal Servis Aracılığı İçin Bir Çok Etmenli Sistem Platformu

Anlamsal veb servisleri ortamında aracılığı gerçekleştirmek için SWSA'nın kavramsal modelinin temel gereksinimlerini yerine getiren bir çok-etmenli sistem platformuna ihtiyaç vardır. FIPA[1] uyumlu böyle bir çok-etmenli sistem platformu, şu ana etmenlerden oluşur: aracı etmen, servis sağlayıcı etmen, servis istemci etmen ve ontoloji

[1] IEEE Foundation for Intelligent Physical Agents (FIPA), http://www.fipa.org/

etmeni. Sonuç olarak, anlamsal veb servisleri ortamında aracılığı gerçekleştirmek için bir çok-etmenli sistem platformu Şekil 1'de görülmektedir.



Şekil 1. Aracılık için bir çok-etmenli sistem platformu

Aracı etmen, bu FIPA uyumlu çok-etmenli sistemde sarı sayfa hizmeti veren dizin servisi (directory facilitator) konumundadır. Aracı etmen, platformdaki anlamsal olarak tanımlanmış servislerin sağlayıcıları ve istemcileri arasında hem keşif hem de arabuluculuk işlevlerini yerine getirir. Aracı etmen, izleyen bölümde detaylı olarak ele alınmaktadır. *Servis sağlayıcı etmenler*, servis sunucu rolü oynarlar. Bu etmenler kendi içsel yeteneklerini anlamsal tanımlamaları aracılığıyla ilan ederler. Aynı zamanda, dışsal veb servislerinin etmen platformuna dahil edilmesini sağlayarak geçit (wrapper) etmen görevini yerine getirirler. Servis sağlayıcı etmenin mimari detayları bu bildirinin kapsamı dışındadır ve [13]'te görülebilir. *Servis istemci etmen*, hedeflerine ulaşmak için diğer etmenlerin servislerine ihtiyaç duyan ve müşteri rolü oynayan bir etmendir. Bir servis istemci etmen herhangi bir servise ihtiyaç duyduğunda, servis isteğini aracı etmene göndererek, uygun servisin/servislerin keşif, seçim ve işletimini aracı etmene havale eder. *Ontoloji etmeni*, platformda kullanılan ontolojilerin ve bunlar arasındaki eşlemelerin depolandığı bir ontoloji havuzu içerir. Ontoloji etmeni, platformun diğer üyelerinin bu ontolojilere kontrollü bir şekilde erişmesini ve sorgulamasını sağlar.

Şekil 1, ayrıca, bu çok-etmenli sistem platformu üzerinde tüm anlamsal servis aracılık senaryosunu da göstermektedir. Servis istemci etmen, aracı etmene bir servis aracılık isteği göndererek süreci başlatır (adım 1). Bu isteği alan aracı etmen, istemci etmenin başarmak istediği içsel hedeflerine potansiyel olarak ulaşmasını sağlayabilecek uygun servis sağlayıcı etmenleri bulur (adım 2). Bundan sonra, aracı etmen keşfedilen bu servis sağlayıcı etmenlerle servisin işletilmesi konusunda uzlaşmaya çalışır (adım 3). Daha sonra, uzlaşılan servis sağlayıcı etmenler ile servisin yürütülmesi gerçekleştirilir (adım 4 ve 5). Son olarak, aracı etmen sonuçları toplar ve servis istemci etmene bir cevap gönderir (adım 6).

4 Aracı Etmen

Genel olarak bahsetmek gerekirse, aracı etmen diğer etmenlerin gereksinimleri ve yetenekleri hakkında bazı bilgileri kullanarak o etmenlere birtakım iletişim kolaylaştırma servisleri sunan bir etmendir. Aracı etmenlerin kullanımı, çok etmenli sistemlerde etmenler arası etkileşimi önemli ölçüde basitleştirir. Buna ek olarak, aracı etmenler bir sistemin dinamik durumlara karşı uyarlanabilir ve sağlam olmasını da sağlar. Çünkü ölçeklenebilirlik ve güvenlik bu etmende sağlanabilir.

Ortamda anlamsal veb servislerinin bulunduğu bir çok-etmenli sistemde aracı etmenin, bu servisleri kullanmak isteyen etmenler (istemci etmenler) ile bu servisleri sağlayan etmenler (sunucu etmenler) arasında hem keşif hem de arabuluculuk işlevlerini yerine getirmesi beklenir. Ayrıca en ucuz, en kaliteli v.b. servisleri bularak, servis sağlayıcılar ile pazarlık yaparak, istemcinin hedefine uygun bir servis olmaması durumunda farklı etmenlerin servislerini birleştirerek; servis istemcinin işlevini kolaylaştırmanın yanında birtakım katma değerler de sağlayabilir.

4.1 Gereksinimler

Anlamsal veb servisleri ortamında bulunan bir aracı etmenin, bu servislerin istemci etmenleri ve sağlayıcı etmenleri arasında hem keşif hem de arabuluculuk işlevlerini yerine getirmesi beklenir. Bunun için, bir aracı etmen aşağıda listelenen temel görevleri yerine getirmelidir:

- Servis sağlayıcıların yetenek ilanlarını kaydetme.
- Servis istemcilerin, bir servis sağlayıcı tarafından yerine getirilmesi gereken servis isteklerini (hedeflerini) yorumlama ve istemcinin gönderdiği girdi parametrelerini uzlaşılacak servisin yürütülmesi sırasında kullanmak üzere saklama.
- Servis istemcinin servis isteğine göre, aday servisleri bulma ve bu servisler arasından en iyisini/iyilerini seçme.
- Seçilen servisin/servislerin sağlayıcısı/sağlayıcıları ile uzlaşma.
- Servis istemci adına, üzerinde uzlaşma sağlanan servisi/servisleri çağırma ve gerekli oldukça servis sağlayıcısıyla/sağlayıcılarıyla etkileşimde bulunma.
- Servis çağrımından elde edilen sonuçları servis istemciye geri gönderme.

Diğer taraftan, yukarıda bahsedilen görevlerin başarımı, beraberinde bazı ek görevleri ve arabuluculuk yetenekleri gerektirir. Dolayısıyla aracı etmen aşağıda açıklanan ek görevleri de yerine getirebilmelidir:

- Uzlaşma aşamasında, potansiyel servis sağlayıcılar ile bir anlaşmaya varıncaya kadar müzakere etme. Müzakereler servis ücreti, servisin kalitesi ve zamanlaması, güvenlik, gizlilik v.b. parametreleri içerebilir.
- Servis kalitesi metriklerinin takibi için servisin işletimini izleme.

Bundan başka, aracı etmen aşağıda belirtilen arabuluculuk yeteneklerine de sahip olmalıdır:

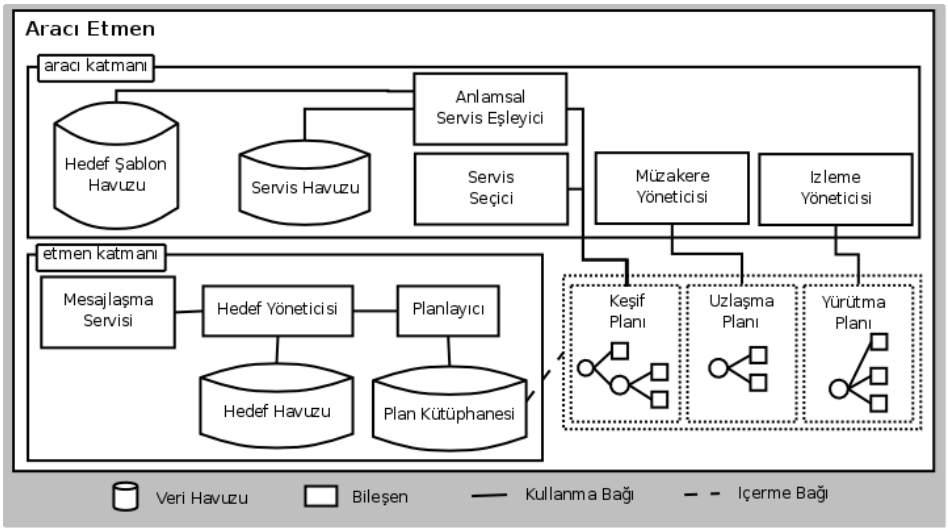
- *Süreç arabuluculuğu*: Servisin işletilmesi sırasında ortaya çıkabilecek olası sorunları giderme:

  - Servis sağlayıcının beklediği ancak servis istemcinin başlangıçta sağlamadığı ek girdiler varsa bunları istemciden tedarik etme. Bu ek girdiler fazladan bir girdi parametresi olabileceği gibi servis istemci tarafından başlangıçta sağlanan girdi parametresinin daha özelleşmişi olan ve servis sağlayıcının beklediği girdi parametresinin bir özniteliği de olabilir.
  - Servis istemcinin sağladığı ancak servis sağlayıcının servisi işletmek için ihtiyaç duymadığı fazladan veriyi (girdiyi) eleme.
  - Servis çağrımı sonucunda oluşan ancak servis istemcinin ihtiyaç duymadığı fazladan veriyi (çıktıyı) eleme.
- *İşlev arabuluculuğu*: Servis istemcinin hedefini karşılayan kayıtlı bir servis bulunamaması durumunda, varolan kayıtlı servisleri kullanarak dinamik servis birleştirimi.

Ayrıca, servis istemci ve sağlayıcının kullandıkları ontolojiler farklı ise, aracının bunların ürettikleri ve/veya tükettikleri veriler arasında çevrim yapması da beklenir (veri arabuluculuğu). Ancak bu çalışmada bahsedilen çok-etmenli sistemde, bu arabuluculuk servis sağlayıcı etmenler tarafından yerine getirildiğinden [13], aracı etmen veri arabuluculuğu yapmamaktadır.

4.2 Aracı Etmenin İçsel Mimarisi

Aracı etmenin mimarisi iki katmandan oluşmaktadır: etmen katmanı ve aracı katmanı (Şekil 2).



Şekil 2. Aracı etmenin içsel mimarisi

Etmen katmanı, çekirdek etmen modüllerini içerir: *Mesajlaşma Servisi*, *Hedef Yöneticisi* ve *Planlayıcı*. *Mesajlaşma Servisi*, etmenin iletişimini yönetir, gelen isteklerden hedefleri çıkartır ve bunları hedef yöneticisine verir. *Hedef Yöneticisi*, etmenin içsel hedeflerini yönetir ve bir hedefe ulaşmak için en iyi planı bulmaktan sorumludur. Bunun için içsel hedef havuzunu kullanır. Aracı etmen, esas olarak *servis kaydetme* ve *servis aracılığı* isimli iki içsel hedefe sahiptir. Servis aracılığı hedefi, her biri SWSA çerçevesinin servis kullanım aşamaları için olmak üzere üç alt-hedef içerir. Bu alt-hedefler için plan kütüphanesinde ayrı ayrı planlar vardır (Şekil 2) ve bu planlar *Planlayıcı* tarafından işletilir. *Keşif planı*, yeteneklerin keşfi için *Anlamsal Servis Eşleyici* modülünü, servis seçimi için ise *Servis Seçici* modülünü kullanarak keşif aşamasını kontrol eder. *Uzlaşma planı,* uzlaşma aşamasını kontrol etmek için kullanılır. Bu plan müzakereleri yönetmek ve zorlukları aşmak için *Müzakere Yöneticisi* modülünü kullanır. *Yürütme planı*, uzlaşılan servisleri çağırır ve *İzleme Yöneticisi* modülünü kullanarak süreci izler. *Planlayıcı*, etmenin kalbidir ve etmenin davranışlarını kontrol eder. Bunun için *Plan Kütüphanesi*'ndeki planları (hedeflere göre) kullanır.

Aracı katmanı, aracılık ile ilgili bileşenleri içerir. Bunlar *servis aracılığı* içsel hedefi tarafından kullanılan *Genel Modüller*, *Hedef Şablon Havuzu* ve *Servis Havuzu*'dur. *Genel Modüller*, uygulama alanına göre ve kullanılan stratejilere göre kolay değiştirilebilen bir mimari sağlarlar. Kısaca bu modüller, *Anlamsal Servis Eşleyici*, *Servis Seçici*, *Müzakere Yöneticisi* ve *İzleme Yöneticisi* modülleridir. *Anlamsal Servis Eşleştirici* modülü, benzer yapıda tanımlanan hedefler ile servis ara yüzlerinin girdi ve çıktı parametreleri arasındaki anlamsal yakınlığı belirlemekle görevlidir. Literatürde anlamsal servis eşleyici olarak yayınlanan bazı çalışmalar mevcuttur [6, 7].

Hedef Şablon Havuzu, platformun çalıştığı alan için önceden tanımlanmış hedef şablonlarını içermektedir. Bu şablonlar, aynı anlamsal veb servislerinin yetenek tanımlarında olduğu gibi, servisin aldığı girdi ve ürettiği çıktı parametrelerinin, platformun alan ontolojisindeki kavramlar kullanılarak tanımlanmasıyla oluşturulmuştur.

Servis Havuzu, platformdaki etmenlerin sundukları ve hedef şablonlarına uygun yapıdaki servislerin anlamsal arayüz tanımlarını içerir. Bu havuz, aracı etmen tarafından sadece arama amaçlı kullanılır. Atomik bir hedef şablonu aracılığıyla gelen bir isteğe uygun bir servis bulunamadığında, var olan kayıtlı servislerin o anda birleştirilerek isteği karşılayan yeni bir servis oluşturulmasına otomatik servis birleştirimi denir. Ancak bu çalışma kapsamında ele alınmamıştır.

4.3 Gerçekleştirim

Önceki bölümlerde gereksinimleri analiz edilen ve içsel mimarisi verilen SWSA çerçevesi tabanlı aracı etmenin bir prototipi gerçekleştirilmiştir. Bu prototip, anlamsal veb ortamında bir çok-etmenli sistem platformu geliştirme altyapısı sunan SEAGENT [14, 15] kullanılarak gerçekleştirilmiştir. SEAGENT[1], anlamsal veb ortamında yaşayan etmenlerin oluşturduğu çok-etmenli sistemler oluşturmak için geliştirilmiş hedef-yönelimli ve anlamsal veb tabanlı bir çok-etmenli sistem çerçevesidir.

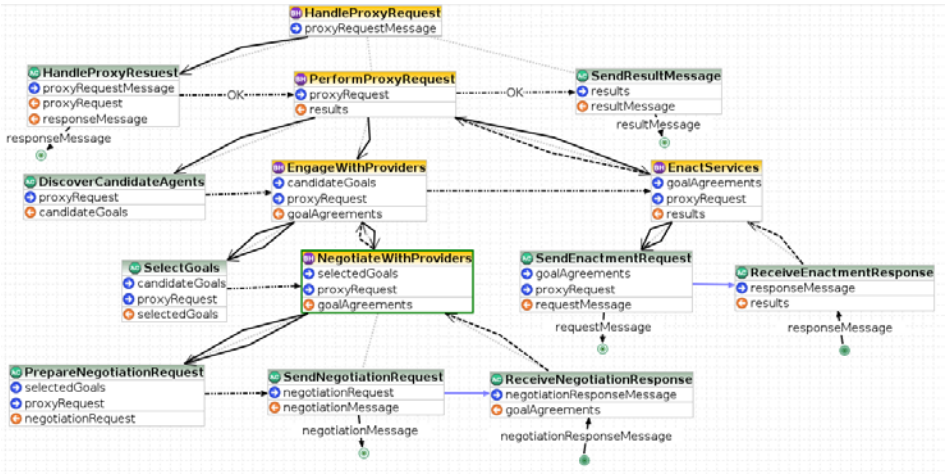
[1] http://etmen.ege.edu.tr/wiki/index.php/Download

Aracı etmenin planlayıcı bileşeni, yine SEAGENT altyapısında bulunan ve HTN (Hierarchical Task Networking) planlama formalizmine dayalı bir planlayıcıdır [16]. Bu planlayıcı[1], çizge tabanlıdır ve Eclipse ortamında görsel bir geliştirme arayüzüne sahiptir [15].

Hedef şablonlarının ve etmenlerin sundukları içsel ya da dışsal servislerin arayüz tanımlamaları için W3C'e sunulan anlamsal veb servis belirtimlerinden OWL-S tercih edilmiştir. Çünkü OWL-S, hem bilimsel çalışmalarda yaygın olarak kullanılmaktadır hem de W3C'nin ontoloji dili standardı olan OWL ile tanımlanmış ontolojilerle ilişkilendirilebilmektedir. Bununla bağlantılı olarak, aracı etmenin ikinci (aracı) katmanında yer alan anlamsal servis eşleştirici olarak OWLS-MX [7] tercih edilmiştir. Bu araç, hem mantık tabanlı çıkarsama yöntemi kullanarak anlamsal yakınlıklara göre hem de çeşitli bilgi erişim yöntemleri kullanarak sözdizimsel yakınlıklara göre servis arama gerçekleştiren melez bir servis eşleyicidir.

Aracı etmenin aracılık için kullandığı etkileşim protokolü, FIPA aracılık protokolünün[2] SWSA servis kullanım aşamalarına uygun olarak genişletilmiş halidir [17]. Bu protokoldeki uzlaşma ve yürütme alt-protokolleri için sadece FIPA istek (request) protokolü gerçekleştirilmiştir. İlerleyen dönemde uzlaşma aşamasında pazarlığa da imkan vermek için FIPA anlaşma (contract net) protokolünün gerçekleştirimi de hedeflenmektedir.

Aracı etmenin plan kütüphanesinde bulunan aracılık planı Şekil 3'de görülmektedir.



Şekil 3. Aracı etmenin aracılık planı

[1] http://seagent.ege.edu.tr/etmen/Distrubition/tr.edu.ege.seagent.htneditor_1.0.1.v20071126.zip
[2] FIPA Brokering Interaction Protocol Specification, http://www.fipa.org/specs/fipa00033/

Özetlemek gerekirse, gerçekleştirilen aracı etmen Bölüm 4.1'de belirtilen temel görevlerin hepsini yerine getirecektir. Ancak aynı bölümde belirtilen ek görevleri (uzlaşma sırasında müzakere ve yürütme sırasında izleme) yerine getirmeyecektir, çünkü aracı etmenin içsel mimarisinin aracı katmanında gösterilen Müzakere Yöneticisi ve İzleme Yöneticisi modülleri gerçekleştirilmemiştir. Yine aynı bölümde bahsedilen arabuluculuk yeteneklerinden bahsetmek gerekirse, gerçekleştirilen aracı etmen, süreç arabuluculuğu ile ilgili belirtilen durumları çözümleyecektir. Ancak, aranan hedefe uygun servis bulunamaması durumunda, varolan servislerin otomatik olarak birleştirilmesi olarak tanımlanan işlev arabuluculuğu, aracı etmen tarafından gerçekleştirilememektedir.

5 Sonuç ve İleriye Dönük Çalışmalar

Bu çalışmada, etmenler alanında yapılan aracılık çalışmalarının ışığında ve SWSA soyut mimari çerçevesine dayalı olarak, anlamsal veb servisleri ortamında bir aracı etmen tasarlanmıştır. Bu tasarımın ana katkısı, SWSA soyut mimari çerçevesine uygun, esnek ve yeniden kullanılabilir bir aracı etmen mimarisi sunulmuş olmasıdır. SWSA soyut mimari çerçevesi, servislerin kullanım süreci için, üç ardışık aşama tanımlamaktadır: keşif, uzlaşma ve yürütme. Her aşama, o sürecin içsel gereksinimlerini karşılamak için farklı etkinlikler içermektedir. Her etkinlik, uygulama gereksinimlerine bağlı olarak farklı şekillerde gerçekleştirilebilir. Bu bağlamda, bu bildiri kapsamında tasarlanan aracı etmen, plan seviyesinde ve modül seviyesinde yeniden kullanım sağlayarak etkinliklerin farklı gerçekleştirimlerinin sisteme kolaylıkla eklenebildiği esnek bir mimari sunmaktadır. SWSA servis kullanım aşamalarını gerçekleştirmek için kullanılan planlar ve bu planların kullandığı yazılım modülleri kolaylıkla değiştirilebilmekte ve yeniden kullanılabilmektedir. Ayrıca, çok etmenli sistemde sunulan servislerin çeşitliliğini artırmak için sadece hedef şablon havuzuna ekleme yapılması yeterlidir.

Tasarlanan aracı etmenin bir prototipi gerçekleştirilmiştir. Bu prototip, temel görevleri yerine getirmekle birlikte bazı eksiklikleri bulunmaktadır. Bundan sonraki dönemde ileriye dönük olarak yapılabilecek çalışmalar aşağıda listelenmiştir:

- Servis sağlayıcıyla uzlaşma aşamasında müzakere ve pazarlık yapılabilmesini sağlayacak Müzakere Yöneticisi modülünün gerçekleştirilmesi, aracı etmenin kullanıcıları için ek katma değer üretebilmesini sağlayacaktır.
- İzleme Yöneticisi modülünün gerçekleştirilmesi, başka servis keşif ve seçim yöntemleri kullanılabilmesini sağlayacaktır.
- Aranan hedefe uygun servis bulunamaması durumunda, varolan servislerin otomatik olarak birleştirilmesini sağlayacak bir Birleştirim Yöneticisi modülünün mimariye eklenmesi ile yine katma değer sağlanabilecektir.

Kaynakça

1. Wong, H. C. ve Sycara, K., "A Taxonomy of Middle-agents for the Internet", In Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000).
2. Paolucci, M., Soudry, J., Srinivasan, N. ve Sycara, K., "A Broker for OWL-S Web Services", Extending Web Services Technologies: The Use of Multi-Agent Approaches, s. 79-98.

3. Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Tanasescu, V., Pedrinaci, C. ve Norton, B., "IRS-III: A Broker for Semantic Web Services Based Applications", In Proceeding of the Fifth International Semantic Web Conference (ISWC 2006), s. 201-214.

4. Klusch, M. ve Sycara, K., "Brokering and Matchmaking for Coordination of Agent Societies: A Survey", Coordination of Internet Agents: Models, Technologies, and Applications, s. 197-224.

5. Burstein, M., Bussler, C., Zaremba, M., Finin, T., Huhns, M., Paolucci, M.,Sheth, A. ve Williams, S., "A Semantic Web Services Architecture", IEEE Internet Computing, 9(5), s. 72–81.

6. Paolucci, M., Kawamura, T., Payne T. R. ve Sycara, K., "Semantic Matching of Web Services Capabilities", In Proceedings of the First International Semantic Web Conference (ISWC 2002).

7. Klusch, M., Fries, B. ve Sycara, K., "Automated Semantic Web Service Discovery with OWLS-MX", In Proceedings of the Fifth İnternational Joint Conference on Autonomous Agents and Multiagent Systems, ACM, s. 915–922.

8. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J. ve Kochut, K., "Quality of service for workflows and web service processes", Journal of Web Semantics, 1:281–308.

9. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J. ve Sheng, Q.Z., "Quality driven web services composition", In Proceedings of the 12th international conference on World Wide Web, ACM Pres, s. 411–421.

10. Sabater J. ve Sierra C., "Reputation and social network analysis in multi-agent systems", In Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), s. 475–482.

11. Sensoy, M., Pembe, F. C., Zirtiloglu, H., Yolum, P. ve Bener, A., "Experience-based service provider selection in agent-mediated e-commerce", International Journal of Engineering Applications of Artificial Intelligence, 20(3), s. 325-335.

12. Vitvar, T., Mocan, A., Kerrigan, M., Zaremba, M., Zaremba, M., Moran, M., Cimpian, E., Haselwanter, T. ve Fensel, D., Semantically-enabled service oriented architecture: Concepts, technology and application. Service Oriented Computing and Applications, 1(2) s.129-154.

13. Gümüs, Ö., Gürcan, Ö., Kardas, G., Ekinci, E. E. ve Dikenelli, O., "Engineering an MAS Platform for Semantic Service Integration based on SWSA", On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, LNCS, 4805:85-94.

14. Dikenelli, O., Erdur, R. C., Gümüs, Ö., Ekinci, E. E., Gürcan, Ö., Kardas, G., Seylan, I. ve Tiryaki, A. M., "SEAGENT: A Platform for Developing Semantic Web based Multi Agent Systems", In AAMAS'05, ACM, s. 1271–1272.

15. Dikenelli, O., "SEAGENT MAS Platform Development Environment", In AAMAS'08, 12-16 Mayıs 2008, (demo için kabul edildi).

16. Ekinci, E. E., Tiryaki, A. M., Gürcan, O. ve Dikenelli, O., "A planner infrastructure for semantic web enabled agents", On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, LNCS, 4805:95-104.

17. Gümüs, Ö., Gürcan, Ö. ve Dikenelli, O., "Towards a Broker Agent in the Semantic Services Environment", Service-Oriented Computing: Agents, Semantics, and Engineering, Lecture Notes in Computer Science, 5006:31-44 (basım için kabul edildi).

Bir Ontoloji Tabanlı Gereksinim Kütüphanesi Yaklaşımı

Görkem Giray[1], Murat Osman Ünalır[2]

[1,2] Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, 35100, Bornova, İzmir
[1] gorkemgiray@mail.ege.edu.tr, [2] murat.osman.unalir@ege.edu.tr

Özet. Bir yazılım geliştirme projesinin çıktısının beklentileri karşılamasının ön koşulu, gereksinimlerin iyi tanımlanmış olmasıdır. Gereksinim mühendisliği sürecinde, kurumun izlediği yöntemlere bağlı olarak çeşitli biçimlerde ve dillerde artifaktlar üretilebilir. Bu artifaktların bir gereksinim kütüphanesinde uygun biçimde saklanması, onların yeniden kullanım amacıyla gereksinim duyulduğu zaman kolayca bulunabilmesi için önemlidir. Gereksinim kütüphanesinin oluşturulması için öncelikle doğal ya da yapay dillerle üretilen artifaktların metamodellerinin oluşturulması ve bu bu artifaktların yarı yapısal bir biçimde temsil edilmeleri gerekmektedir. Daha sonra, bu artifaktlar ile alan ontolojileri arasındaki eşleştirmeler şema eşleştirme algoritmaları kullanılarak yarı otomatik olarak yapılabilir. Anlamsal web teknolojilerinin temelini oluşturan ontolojilerin gereksinim kütüphanesinde kullanımı, bilgisayarlara anlamsal seviyede bilgi işleyebilme olanağı sunacaktır. Elde edilen ontoloji tabanlı gereksinim kütüphanesinde, anlamsal seviyede arama yapılarak sözdizimsel seviyedeki aramaya göre daha başarılı sonuçlar elde edilecektir. Bunun yanında, alan ontolojileri kullanılarak yapılacak çıkarsama işlemleri sonucunda, gereksinim artifaktlarının kaliteleri hakkında bilgi sahibi olmak ve bu artifaktların kalitelerinin arttırılması için bazı öneriler elde etmek mümkündür.

1 Giriş

Gereksinimlerin, yazılım geliştirme sürecinin başarısındaki etkisi büyüktür. Gereksinimlerin iyi tanımlanmadığı bir yazılım geliştirme sürecinin çıktısının başarılı olması beklenemez. Gereksinim mühendisliği sürecinde karşılaşılan zorluklar çeşitli araştırmacılar tarafından irdelenmiştir. Gereksinim mühendisliğinin temel zorluğu, yapısal olmayan gerçek dünyadaki problemleri, yapısal bilgisayar dünyasında sıfır ve bir kesinliğinde ifade etmek ve bu problemlerin çözümleri için gerekli bilgileri sağlamaktır. Bu süreçte rol alan paydaşlara destek vermek amacıyla gereksinim mühendisliği sürecinde kullanılabilecek birçok yöntem ve araç geliştirilmiştir. Gereksinimlerin yeniden kullanımını destekleyen yöntemler ve araçlar da bu kapsamda değerlendirilir.

Anlamsal web çalışma alanının gerek akademik gerekse iş dünyasında popüler olmasıyla birlikte, uzun zamandır yapay zeka çalışma alanında kullanılan ontolojiler daha geniş bir kitlenin dikkatini çekmiştir. Ontolojiler, bilginin anlamının bilgisayarların işleyebileceği biçimde temsil edilmesini sağlayarak, bilginin daha zekice işlenebilmesini sağlayabilmektedir. Buradaki "zekice işleme" kavramının insan davranışlarını taklit etme anlamına gelmediği anlamsal web'in fikir babası olan Tim Berners-Lee tarafından açıkça belirtilmiştir [1]. Dolayısıyla ontolojilerin kullanımı

beraberinde, bilgisayara daha fazla işlenebilir bilginin uygun biçimde sunulmasını gerektirmektedir. Buna karşılık, bilgisayarın uygun biçimde tanımlanmış bilgi üzerinde yapacağı çıkarsama ile saklı (yani açık olarak ifade edilmeyen) bilgiye ulaşılması mümkün olacaktır.

Literatürde, gereksinim mühendisliği sürecinde üretilen artifaktların ontolojilerdeki kavramlarla eşleştirildiği ve bu eşleşmelerden yola çıkılarak çeşitli çıkarsama işlemlerinin yapıldığı çalışmalar bulunmaktadır [2], [3], [4], [5]. Ayrıca gereksinimlerin ve çeşitli yazılım bileşenlerinin saklandığı kütüphanelerin örnekleri de vardır [6]. Yine bu bildiri kapsamındaki çalışmaya katkıda bulunabilecek, yazılım bileşenlerinin ontoloji tabanlı yeniden kullanım kütüphanelerinde saklandığı projeler bulunmaktadır [7], [8].

Bu çalışmanın amacı, ontolojilerin gereksinim kütüphanelerinin oluşumunda kullanıldığı bir yöntem geliştirmektir. Bu yöntem kapsamında, paydaşlar tarafından üretilen gereksinim artifaktlarının alan ontolojileri ile eşleştirilmesi işleminin şema eşleştirme algoritmaları kullanılarak yarı otomatik gerçekleştirilmesi ve elde edilen eşleştirmelerin uygun biçimde saklanmasıyla birlikte anlamsal seviyede arama yapılabilecek bir gereksinim kütüphanesinin elde edilmesi hedeflenmektedir. Ayrıca, alan ontolojileri kullanılarak yapılacak çıkarsama işlemleri sonucunda gereksinim artifaktlarının kaliteleri üzerine paydaşlara geri bildirim verilmesi ve kalitenin arttırılması için önerilerde bulunulması mümkün olacaktır. Bu bildirinin yazıldığı zamana kadar, çalışmanın çözüm bulmak istediği problem tanımlanmıştır, problemin çözümü için bir çerçeve geliştirilmiştir ve beklenen faydalar ortaya konulmuştur. Bundan sonraki adımda, ontoloji tabanlı gereksinim kütüphanesinin geliştirilmesi için bir yöntem önerilecektir ve bu yöntem kullanılarak örnek bir uygulama geliştirilecektir. Daha sonra da geliştirilen örnek uygulama üzerinde yöntemin başarısı sınanacaktır.

Bildirinin ikinci bölümünde, gereksinim mühendisliği çalışma alanı hakkında kısaca bilgi verilmiştir ve gereksinimlerin yeniden kullanımının getirebileceği faydalar belirtilmiştir. Üçüncü bölümde, bu bildiri kapsamında yapılacak çalışma anlatılmıştır. Dördüncü bölümde, bu bildiri kapsamındaki çalışmaya katkıda bulunabilecek ya da bu çalışma ile kıyaslanabilecek literatürdeki yayınlar sunulmuştur. Beşinci bölümde, bildirinin sonuçları özetlenmiştir.

2 Gereksinim Mühendisliği

Bir sistemin, sunması gereken hizmetleri ve bu hizmetleri hangi kısıtlar altında çalışırken sunacağının belirlenmesi sürecine gereksinim mühendisliği adı verilmektedir. Bu tanımdaki mühendislik sözcüğü, geliştirilecek sistemin tanımlanması için sistemli bir sürecin uygulanacağını belirtmek amacıyla kullanılır [9].

Gereksinim mühendisliğinde yaşanan zorluklar, gerek teorik gerekse pratik açıdan birçok araştırmacı tarafından irdelenmiştir. Bu konuyu teorik açıdan inceleyen araştırmacıların ortak düşüncesi, gereksinim mühendisliğinin zorluklarının temelinde, biçimsel olmayan gerçek dünyanın biçimsel olarak modellenmesi çabasının bulunduğu yönündedir.

Kotonya ve Sommerville, ideal bir gereksinim mühendisliği sürecinin varlığından söz edilebilmesinin mümkün olmadığını dile getirmektedir [10]. Her kurum, geliştirdiği

sisteme, kurum kültürüne, ekibi oluşturan kişilerin yetenek ve tecrübelerine göre kendi sürecini geliştirmelidir [10]. Bir gereksinim mühendisliği sürecinde farklı gereksinim artifaktları üretilebilir. Bu artifaktlar, gereksinimlerin farklı yönlerini, farklı yöntemlerle ve kabullerle ortaya koyar.

Yazılım mühendisliğinde yeniden kullanım, mevcut yazılım artifaktlarının yeni bir yazılım sisteminin inşasında kullanılması olarak tanımlanır. Yeniden kullanılabilecek artifakt tipleri sadece kod ile sınırlı değildir. Gereksinim belgeleri, gereksinim belirtimleri, tasarımlar, bileşenler, kod, test senaryoları gibi yazılım geliştirme sürecinin çeşitli aşamalarında üretilen ara ürünler, yeni bir yazılım sisteminin inşasında yeniden kullanılabilir [11]. Literatürdeki yeniden kullanım üzerine yapılmış yayınlar daha çok kodun yeniden kullanımı üzerinedir. Yeniden kullanımın, yazılım geliştirme sürecinin başlarında, yani gereksinimlerin oluşturulduğu safhada uygulanabileceği yönünde öneride bulunan ve bu konuda yapılan projeleri tanıtan yayınlar da bulunmaktadır [12], [13], [14], [15], [16], [17]. Bu çalışmalara göre, gereksinim artifaktlarının yeniden kullanımının faydaları şu şekilde özetlenebilir:

- Tekrar kullanılan gereksinim artifaktlarının, baştan yazılan gereksinim artifaktlarına göre daha olgun ve güvenilir olma olasılığı yüksektir [12].
- Yazılım geliştirme sürecinde üretilmesi gereken her artifaktın (gereksinim artifaktları da dahil), yeniden kullanım stratejisi doğrultusunda mevcut bir artifaktın aynen ya da değiştirilerek sistemle bütünleştirilmesiyle üretilmesi ekonomik kazançlar sağlayacaktır. Özellikle büyük ölçekli yazılım geliştirme projelerinde, yeniden kullanılan artifakt sayısı arttıkça, elde edilen ekonomik kazançlar artacaktır [12].
- Yazılım geliştirme sürecinde yeniden kullanım ne kadar erken başlarsa, elde edilecek kazanımlar da o kadar artacaktır. Bu durumda yeniden kullanımın gereksinim artifaktlarının üretimiyle başlaması sonucunda elde edilecek fayda, kod seviyesinde yeniden kullanım sonucunda elde edilecek kazanımlara göre, diğer koşullar aynı kalmak şartıyla, daha fazla olacaktır. Tabii ki bu faydanın elde edilebilmesinin arkasında, gereksinim artifaktlarının yeniden kullanımıyla birlikte, bu artifaktları gerçekleyen tasarımların ve kodun da yeniden kullanılacağı varsayımı yatmaktadır [13], [15], [17].

Gereksinim mühendisliği sürecinde üretilen artifaktlar biçimsel, yarı biçimsel ya da biçimsel olmayan bir yapıya sahip olabilir. Gereksinimlerin tamamen biçimsel dillerle ifade edildiği yöntemler literatürde mevcuttur. Bu biçimsel dillerin hiçbirisi tüm yazılım sistemlerinin geliştiriminde kullanılabilecek kadar genel amaçlı değildir. Dolayısıyla, birçok yayında da [13], [14] belirtildiği gibi, gereksinim artifaktlarında doğal dil kullanımından tamamen vazgeçmek mümkün görünmemektedir. Bu doğrultuda, bildiri kapsamında ortaya konulan çerçeve, doğal dil cümleleri de içeren farklı gereksinim artifaktlarının varlığını göz önüne almaktadır.

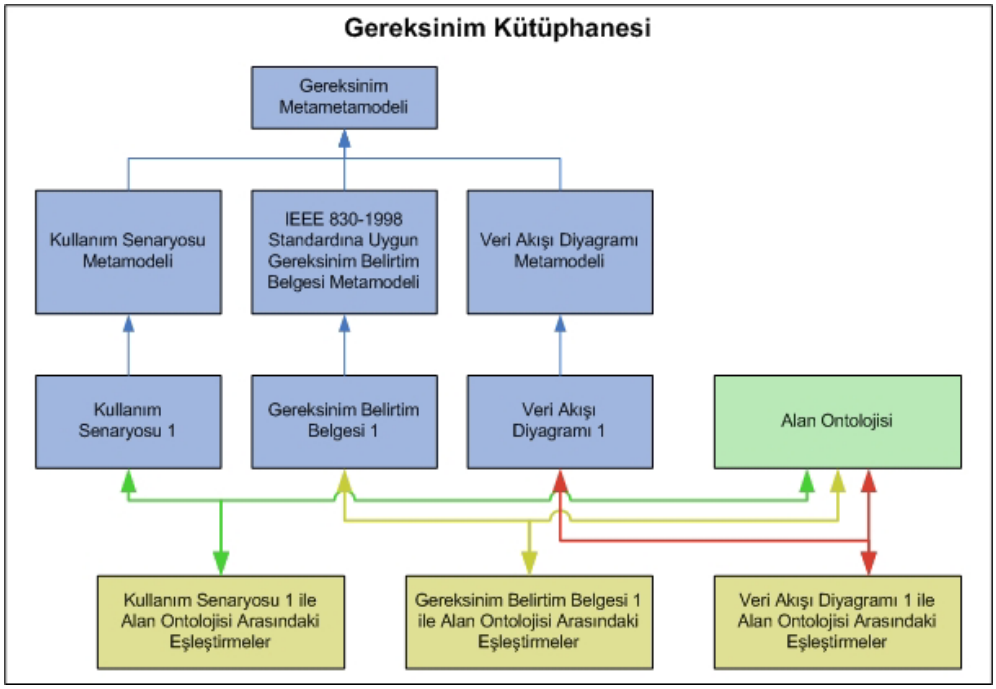
3 Ontoloji Tabanlı Gereksinim Kütüphanesi İçin Kavramsal Çerçeve

Gereksinim artifaktları, bir kurumun iş yapma biçimini, farklı kapsamlarda (kurumun tüm süreçlerini, kurumun bir bölümünün tüm süreçlerini, kurumu ilgilendiren bir süreci, vs.) mümkün olduğu kadar net ifade eder. Bu açıdan bakıldığında gereksinim artifaktları, kurum ölçeğindeki değişik yazılım geliştirme projelerinde, aynı alanda hizmet veren kurumlar için geliştirilen farklı projelerde ya da bir yazılım geliştirme şirketinin yazılım ürün hattı kapsamındaki projelerinde kullanılabilir. Yazılım geliştirme sürecinde üretilen artifaktların yeniden kullanılabilmesi için bulunabilir, anlaşılabilir ve kullanılabilir olması önemlidir. Bir yeniden kullanım kütüphanesi, yeniden kullanılabilir artifaktların saklandığı bir havuzdan, kütüphanede arama yapılmasına izin veren bir arayüzden ve artifaktların saklanması için kullanılan bir temsil yönteminden oluşur [18]. Gereksinim artifaktlarının alan ontolojileriyle eşleştirilerek saklanması, bu artifaktların anlamlarının bildirimsel olarak temsil edilmesini sağlayacaktır. Bunun sonucunda, sadece sözdizimsel seviyede değil aynı zamanda anlamsal seviyede de arama yapma olanağı elde edilerek arama işlemlerinin başarısı arttırılacaktır.

Bir yazılım geliştirme projesi kapsamında üretilen gereksinim artifaktları, bir kurum için, bilgi yönetimi kapsamında değerlendirilmesi gereken önemli varlıklardır. Uren ve arkadaşları, bilgi yönetimi için anlamsal etiketleme sistemlerinin sahip olması gereken özellikleri, standart formatlar, kullanıcı merkezli ve işbirliği ile oluşturulan tasarım, ontoloji desteği, heterojen belge formatı desteği, belge evrimi, etiket saklama ve etiketleme işleminin otomatikleştirilmesi olmak üzere yedi başlık altında toplamışlardır [19]. Bildiri kapsamında geliştirilen yöntemde aşağıdaki başlıklar üzerinde yoğunlaşılacaktır:

- Standart formatlar: Ontolojiler, geniş kesimler tarafından kabul edilmiş dillerle (RDF Schema, OWL) temsil edilecektir.
- Ontoloji desteği: Geliştirilecek yöntem, ontolojilerin gereksinim kütüphanesi oluşumunda kullanımını kapsayacaktır.
- Heterojen belge formatı desteği: Gereksinim kütüphanesinde farklı gereksinim artifaktlarının saklanması desteklenecektir.
- Etiketleme işleminin otomatikleştirilmesi: Gereksinim artifaktlarının alan ontolojileriyle eşleştirme işlemleri, şema eşleştirme algoritmaları ile yapılacaktır.
- Etiket saklama: Elde edilen eşleştirmeler kütüphanede saklanacaktır.

Çalışma kapsamında geliştirilecek yöntemin uygulanması sonucunda, Şekil 1'de ana bileşenleri görülen bir gereksinim kütüphanesi elde edilmesi hedeflenmektedir. Bu yöntem kapsamında, kütüphanede saklanacak gereksinim artifaktlarının metametamodeli ve her bir gereksinim artifakt tipinin metamodeli tanımlanacaktır. Gereksinim artifaktları doğal dil ifadeleri içereceği için, doğal dil işleme tekniklerinden faydalanılacaktır. Metamodeller ise artifaktları yarı yapısal biçime getirmeyi ve böylece otomatik eşleştirme işleminin başarısını arttırmayı hedeflemektedir.



Şekil 1. Gereksinim kütüphanesinin ana bileşenleri

Metamodellere uygun olarak üretilen gereksinim artifaktlarının alan ontolojileri ile yarı otomatik olarak eşleştirilmesini sağlayacak şema tabanlı, örnek tabanlı ve yeniden kullanım yönelimli eşleştirme [20] algoritmaları geliştirilecektir. Şema tabanlı eşleştirme algoritmaları, şema bilgisini temel alarak eşleştirme yapmaya çalışır. Kullanılan şema tanımlama dilinin temsil yeteneğine bağlı olarak şema bilgisi, şema elemanlarının özellikleri (isim, açıklama, veri tipi, kısıt, vs.) ve şema elemanları arasındaki ilişkileri (referans kısıtları, is-a, part-of veya içerme ilişkileri gibi) içerebilir. Örnek tabanlı eşleştirme algoritmaları ise, şema elemanları arasındaki ilişkileri tespit etmek için örnek verisini inceler. Yeniden kullanım yönelimli eşleştirme algoritmaları, şema ve örnek bilgilerine ek olarak, ikincil bilgi kaynaklarını (genel veya alana özgü sözlükler, eş anlamlılar sözlüğü, vs.) kullanır [20]. Bu eşleştirme algoritmaları, ontoloji tabanlı gereksinim kütüphanesinin oluşturulması için özelleştirilecek ve doğal dil işleme teknikleriyle zenginleştirilecektir.

Gereksinim artifaktları ile alan ontolojileri arasındaki eşleştirmelerin uygun biçimde saklanmasıyla ve yarı yapısal gereksinim artifaktları üzerinde daha başarılı arama işlemleri yapılmasıyla gereksinimlerin yeniden kullanımı desteklenecektir. Bu çalışmanın ilerideki aşamalarında, gereksinim artifaktları üzerinde, alan ontolojilerinden yola çıkılarak çeşitli çıkarsama işlemleri yapılarak, gereksinimlerin kaliteleri ölçülebilir ve iyileştirme fırsatları için paydaşlara geri bildirim verilebilir.

Şekil 1'de görüldüğü gibi, gereksinim kütüphanesinde saklanabilecek kullanım senaryoları, veri akış diyagramları, gereksinim belirtim belgelerinin içeriği üzerine önerilerde bulunan IEEE 830-1998 standardının [21] metamodelleri, gereksinim metametamodeline uyumlu olarak tanımlanır. Yöntem kapsamında paydaşlar tarafından geliştirilen artifaktlar ise, metamodellerin birer somutlaşmış örneğidir. Oluşturulan artifaktlar, kütüphanede bulunan ya da kütüphaneye eklenen alan ontolojilerindeki kavramlar ile eşleştirme algoritmaları aracılığıyla eşleştirilir. Bu otomatik eşleştirme işleminin sonucu paydaşlar tarafından kontrol edilir ve eksik ya da yanlış eşleştirmeler düzeltilir. Eşleştirmeler, ontoloji tabanlı arama yapılabilecek şekilde kütüphanede saklanır. Gereksinim mühendisliği sürecinde üretilen farklı artifaktlar için metamodeller geliştirilerek kütüphanenin saklayabileceği artifakt çeşidi arttırılabilir. Benzer şekilde, kütüphanenin içerdiği alan ontolojileri genişletilebilir ya da kütüphaneye yeni alan ontolojileri eklenebilir.

4 İlgili Çalışmalar

Literatürde, gereksinim mühendisliği etkinliklerinde ve yeniden kullanım kütüphanelerinde ontolojilerden faydalanılan çeşitli çalışmalar bulunmaktadır. Bu çalışmaları, bu bildiri açısından önemli sayılan özellikleri doğrultusunda sınıflandırmak mümkündür.

İlk sınıftaki çalışmaları, bu bildirideki gibi, mevcut gereksinim artifaktlarını alan ontolojileriyle eşleştiren ve bu eşleştirmeler üzerinden paydaşlar için faydalı çıkarsamalar yapan çalışmalar oluşturmaktadır.

Kaiya ve Saeki, alan ontolojilerinin gereksinimlerin elde edilmesi safhasında alan bilgisinin temsili olarak kullanılması için bir yöntem önermişlerdir [2]. Daha sonra bu çalışmaları kapsamında, amaca yönelik ve ontoloji güdümlü gereksinim elde etme yöntemi geliştirmişlerdir [3]. Bu yöntemde, gereksinimlerin sistematik olarak elde edilmesi ve tanımlanabilmesi için, gereksinim listelerinin, ontolojilerin ve gereksinim listeleri ile ontolojiler arasındaki anlamsal eşleştirmelerin yapısı biçimsel olarak belirlenmiştir. Yöntemde, gereksinim listeleri ile ontolojiler arasındaki eşleştirmeler paydaşlar tarafından yapılmaktadır.

Lee ve Zhao'nun önerdikleri ontoloji tabanlı gereksinim elde etme ve çözümleme yöntemi üç adımdan oluşmaktadır [4]. Birinci adımda, alan terimleri ve gereksinimleri belirtilir. İkinci adımda, gereksinimler alt gereksinimlere ayrıştırılır. Bunun için öznel ayrıştırma yöntemi (subjective decomposition) kullanılır. Üçüncü adımda ise gereksinimler önce işlevsel ve işlevsel olmayan daha sonra da en alt seviyede atomik gereksinimlere ayrıştırılır. Daha sonra bu en alt seviyedeki gereksinimler ontolojideki kavramlarla ilişkilendirilir. Bu yöntem, ontolojileri kullanarak paydaşların gereksinim elde etme evresindeki etkinliklerini arttırmayı amaçlamaktadır. Bunun yanında ontoloji ile eşleştirilen gereksinimlerin tam olup olmadığı ve çelişkiler içerip içermediğini çıkarsama yaparak bulmayı amaçlamaktadır. Bir önceki yöntemdeki gibi, gereksinim artifaktları ile alan ontolojileri arasındaki eşleştirmeler paydaşlar tarafından yapılmaktadır.

SoftWiki projesinin amacı, yazılım geliştirme süreçlerinde çalışan tüm paydaşların özellikle yazılım gereksinimlerinin oluşturulmasına yönelik olarak birlikte çalışmalarını

desteklemek ve kolaylaştırmaktır [5]. Bu amacı gerçekleştirmek için, çok sayıda ve fiziksel olarak farklı yerlerde bulunan paydaşların yazılım gereksinimlerini toplayabilmeleri, bunları anlamsal olarak zenginleştirebilmeleri, sınıflandırabilmeleri ve birleştirebilmeleri gerekmektedir. Oluşturulan çözüm, bilgi temsili için anlamsal web standartları üzerine temellendirilmiştir [5]. Gereksinim mühendisliği sürecini desteklemek amacıyla SWORE (SoftWiki Ontology for Requirements Engineering) ontolojisi geliştirilmiştir. Geliştirilen artifaktların, hem SWORE ontolojisindeki hem de alan ontolojisindeki kavramlarla eşleştirme işlemi paydaşlar tarafından yapılmaktadır.

Bu çalışma ile ilgili ikinci bir sınıf oluşturabilecek çalışmalar ise ontolojilerin bilgi yönetiminde kullanımı üzerine temellenmektedir. Bu çalışmalara bir örnek olarak, Wu ve arkadaşlarının ontoloji tabanlı yazılım kütüphanelerinin metin madenciliği ile inşası verilebilir [22]. Bu çalışmada, doğal dil ile üretilmiş gereksinim belgelerinden metin madenciliği teknikleri kullanılarak ontolojilerin oluşturulması ve bu ontolojilerin gereksinim belgelerinde daha verimli arama yapmak için kullanıldığı bir yöntem önerilmektedir. Bu çalışmadaki yöntemde, doğal dil kullanılarak üretilmiş gereksinim belgeleri hedeflenmiş ve ontolojilerin de bu yöntem içinde inşa edileceği öngörülmüştür. Bu çalışmadan farklı olarak, çeşitli gereksinim artifaktlarının desteklenmesi ve otomatik şema eşleştirme yöntemlerinin kullanımı bizim çalışmamızı Wu ve arkadaşlarının çalışmasından ayırmaktadır.

Üçüncü sınıftaki çalışmalar ise bu bildiri kapsamında anlatılan yaklaşımı, yazılım bileşenleri kütüphanelerine uygulamaktadır. Bu çalışmalarla, bildirideki yaklaşımın benzer yönleri olduğu gibi farklı yönleri de bulunmaktadır.

Happel ve arkadaşlarının KontoR isimini verdikleri yeniden kullanım yaklaşımı, ontolojileri kullanarak yeniden kullanım kütüphanelerinin yazılım geliştirme projelerine sağladıkları katkıları arttırmayı hedeflemektedir [7]. Bunun için metaveri tabanlı yeniden kullanım yaklaşımlarını artifakt ya da kullanıcı bağlam bilgisini kullanarak geliştirme yolunu seçmişlerdir. Yaklaşımın desteklendiği aracın mimarisinin bileşenleri, metaveri kütüphanesi, bilgi tabanı, çıkarsama motoru ve SPARQL arayüzü olarak tanımlanmıştır. Metaveri kütüphanesinde, yaklaşım kapsamında önerilen metaşemaya uygun metaveri saklanmaktadır. Metaveri, alan ontolojileriyle genişletilebilen bir yüzeysel ontoloji ile zenginleştirilmektedir. Böylece bağlam bilgisini temsil eden alan ontolojileri bilgi tabanında saklanabilmektedir ve çıkarsama motoru aracılığıyla bu bilgi üzerinde çıkarsama yapılabilmektedir. Yeniden kullanım kütüphanesinde, anahtar kelime tabanlı aramanın yanında SPARQL ile temsil edilen anlamsal sorgularla da arama yapılabilmektedir.

Antunes ve arkadaşları, yazılım geliştirme bilgisinin yeniden kullanımında ontolojileri kullanan "anlamsal yeniden kullanım sistemi"ni önermişlerdir [8]. Sistem, temsil ontolojisi ve alan ontolojisi olmak üzere iki tip ontoloji kullanmaktadır. Temsil ontolojisi yeniden kullanılabilir yazılım geliştirme birimlerininin yapısını tanımlamaktadır. Alan ontolojisi ise yazılımın geliştirdiği alandaki bilgiyi temsil etmektedir. Temsil ontolojisine uygun olarak oluşturulmuş yeniden kullanılabilir yazılım bileşenleri, alan ontolojisindeki kavramlarla eşleştirilerek saklanmaktadır. Böylece yeniden kullanım sisteminde anlamsal seviyede arama yapılabilmektedir.

Bu bildirideki yaklaşıma paralel olarak, Happel'in [7] ve Antunes'in [8] çalışmaları da, yazılım geliştirme sürecinde üretilen bileşenleri, önceden tanımlanmış metamodellere uygun metaveri ile betimlemektedir. Bu betimlemeler alan ontolojileriyle eşleştirilmektedir. Happel'in ve Antunes'in çalışmalarından farklı olarak bu bildiride, gereksinim artifaktlarının yeniden kullanımı üzerine vurgu yapılmaktadır. Ayrıca, metaverinin alan ontolojilerindeki kavramlarla eşleştirilmesi işleminin, eşleştirme algoritmaları kullanılarak yarı otomatik yapılması önerilmektedir.

5 Sonuçlar

Anlamsal web'in amacının, bilgisayarlara, insanın kullandığı doğal dili anlama yeteneği kazandırmak değil; insanların daha fazla çaba sarf ederek, bilgisayarlara biçimsel olarak ifade ettikleri veri üzerinde işlemler yaparak, bundan çeşitli faydalar elde etmek olduğu Tim Berners-Lee tarafından belirtilmiştir [1]. Ontolojilerin, gerek İnternet uygulamalarında gerekse kurum içi uygulamalarda kullanılması, bu uygulamalara bilgisayar tarafından işlenebilir bilginin sağlanma gerekliliğini ortaya çıkaracaktır. Bu bağlamda, ontolojilerin gereksinim kütüphanelerinde kullanımının sağlayacağı faydayı en üst seviyeye çıkarmak için, ontolojilerle yarı yapısal gereksinim artifaktlarının eşleştirme işleminin mümkün olduğu kadar otomatik olarak yapılması önemlidir. Bu bildiri kapsamında, bu faydayı en üst düzeye çıkarmak için özelleştirilmiş şema eşleştirme yöntemlerinin doğal dil işleme teknikleriyle birlikte kullanımı önerilmektedir. Ayrıca gereksinim kütüphanesinin farklı artifakt yapılarını ve anlamsal seviyede arama yapılabilmesini desteklemesinin önemi vurgulanmaktadır.

Yeniden kullanım hedefiyle gereksinimlerin üretilmesi, beraberinde, belli yöntemlerin kullanımını ve belli kurallara uygunluğun sağlanmasını getirecektir. Dolayısıyla, hiçbir yöntemin, metodolojinin ya da sürecin tüm yazılım geliştirme projelerinde kullanılabilmesi mümkün değildir. Bu bağlamda her kurum, geliştirdiği sisteme, kurum kültürüne, ekibi oluşturan kişilerin yetenek ve tecrübelerine göre kendi gereksinim mühendisliği sürecini geliştirmelidir. Bu sürecin içinde ontoloji tabanlı yöntemlerin uygulanıp uygulanmaması da sürecin tasarımını etkileyecek etkenlere bağlıdır. Sürecin çeviklik derecesi bu kullanımı etkileyecek önemli girdilerden birisidir. Her yinelemede, çalışan bir işlevin çıktı olarak üretildiği, yineleme sürelerinin kısa olduğu ve gereksinim belgelemeye çok fazla zamanın ayrılamadığı çeviklik derecesi yüksek projelerde ontoloji tabanlı yöntemlerin kullanılma olasılığı gayet düşükken, tüm gereksinimlerin detaylı olarak belgelenmesi gereken çeviklik derecesi düşük projelerde bu olasılık yüksek olacaktır.

Kaynakça

1. Berners-Lee, T., "What the Semantic Web can represent", http://www.w3.org/DesignIssues/RDFnot.html (01.04.2008), 1998.
2. Kaiya, H. ve Saeki, M., "Using Domain Ontology as Domain Knowledge for Requirements Elicitation", Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06): IEEE Computer Society, 2006.
3. Shibaoka, M., Kaiya H. ve Saeki, M., "GOORE: Goal-Oriented and Ontology Driven Requirements Elicitation Method", Advances in Conceptual Modeling – Foundations and Applications, 2007, s. 225-234.

4. Lee, Y. ve Zhao, W., "An Ontology-Based Approach for Domain Requirements Elicitation and Analysis", Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences - Volume 2 (IMSCCS'06): IEEE Computer Society, 2006.

5. Auer, S., Jungmann, B. ve Schönefeld, F., "Semantic Wiki Representations for Building an Enterprise Knowledge Base", Reasoning Web, 2007, s. 330-333.

6. Mili, A., Mili, R. ve Mittermeir, R. T., "A survey of software reuse libraries", Annals of Software Engineering, vol. 5, 1998, s. 349-414.

7. Happel, H.-J., Korthaus, A., Seedorf, S. ve Tomczyk, P., "KOntoR: An Ontology-enabled Approach to Software Reuse", 18th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE '06), San Francisco, CA, 2006.

8. Antunes, B., Seco, N. ve Gomes, P., "Using Ontologies for Software Development Knowledge Reuse", Progress in Artificial Intelligence, 2007, s. 357-368.

9. Sommerville, I., Software engineering (5. baskı): Addison Wesley Longman Publishing Co., Inc., 1995.

10. Kotonya, G. ve Sommerville, I., Requirements Engineering: Processes and Techniques: John Wiley & Sons, Inc., 1998.

11. Krueger, C. W., "Software reuse", ACM Comput. Surv., Volume 24, 1992, s. 131-183.

12. Lam, W., McDermid, T. A. ve Vickers, A. J., "Ten steps towards systematic requirements reuse", Proceedings of the Third IEEE International Symposium on Requirements Engineering, 1997.

13. Keepence, B., Mannion, M. ve Smith, S., "SMARTRe requirements: writing reusable requirements", Proceedings of the International Symposium and Workshop on Systems Engineering of Computer Based Systems, 1995.

14. Mannion, M., Keepence, B., Kaindl, H. ve Wheadon, J., "Reusing single system requirements from application family requirements", Proceedings of the International Conference on Software Engineering, 1999.

15. Neighbors, J. M., "An assessment of reuse technology after ten years", Proceedings of the Third International Conference on Software Reuse: Advances in Software Reusability, 1994.

16. Lam, W., "Scenario reuse: a technique for complementing scenario-based requirements engineering approaches", Proceedings of Asia Pacific and International Computer Science Conference – APSEC'97 and ICSC'97, 1997.

17. Paredes, C. ve Fiadeiro, J. L., "Reuse of requirements and specifications: a formal framework" SIGSOFT Softw. Eng. Notes, Volume 20, 1995, s. 263-266.

18. Frakes, W. B. ve Kyo, K., "Software reuse research: status and future", IEEE Transactions on Software Engineering, Volume 31, 2005, s. 529-536.

19. Uren, V., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E. ve Ciravegna, F., "Semantic Annotation for Knowledge Management: Requirements and a Survey of the State of the Art", Journal of Web Semantics: Science, Services and Agents on the World Wide Web, Volume 4, 2006, s. 14–28.

20. Do, H.-H., "Schema Matching and Mapping-based Data Integration", Department of Computer Science, Doktora tezi, Leipzig, Germany: Universität Leipzig, 2006.

21. “IEEE recommended practice for software requirements specifications”, IEEE Std 830-1998, 1998.

22. Wu, Y., Siy, H., Zand, M. ve Winter, V., “Construction of Ontology-Based Software Repositories by Text Mining”, Computational Science – ICCS 2007, 2007, s. 790-797.

Ontoloji Tabanlı Model Dönüşüm Yöntemlerinin İncelenmesi

Alpay Doruk[1]

[1] İzmir Yüksek Teknoloji Enstitüsü, Bilgisayar Mühendisliği Bölümü, 35437, Urla, İzmir
[1] alpaydoruk@iyte.edu.tr

Özet. Bu makalede, Ontology Management Group (OMG) tarafından tanımlanan Model Güdümlü Mimari (Model Driven Architecture (MDA)) kapsamındaki model dönüşümlerinin ontolojiler kullanılarak gerçekleştirilmesi konusu incelenmiştir. Model dönüşümünün ontolojiler ile gerçekleştirilmesinin olası faydaları, anlamsal olarak birbiriyle tam olarak eşleşen modellerin ortaya çıkmasıdır. Bu kapsamda öncelikle model dönüşümünün ne olduğu ve literatürde kullanılan bazı model dönüşüm dilleri konusunda bilgi verilecektir. Makalenin ilerideki bölümlerinde ise ontoloji tabanlı model dönüşümü, dönüşüm bileşenlerinin neler olabileceği ve ontoloji tabanlı model dönüşümünün ne gibi adımlardan oluşabileceği konusunda bilgi verilecektir. En sonda ise literatürde bu konuda yapılan çalışmalar ile ilgili kısaca bilgi verilecektir.

1 Giriş

Bu bildiri kapsamında öncelikle Ontology Management Group (OMG)'nin Model Güdümlü Mimarisi (Model Driven Architecture (MDA)) [1][2][3] incelenmektedir. "MDA'da mimari modellere dayanmaktadır. Model, bir sistemin geliştirilme, çalışma ya da bakım aşamalarındaki belirgin bir görünümüdür. Her model belirgin bir üst-modele bağımlıdır ve spesifik bir üst-model dili ile yazılmalıdır. Bir üst-model, bir sistemin ilintili görünümlerini çıkaran ve diğer detayları ayıklayan bir filtre gibi davranmaktadır. Üst-üst-modeller ise üst-modelleri geliştirmek için gereken bir dil tanımlar. Bir üst-üst-model en azından üç kavrama (varlık (entity), ilişki (association) ve paket (package)) ve bir ilkel tipler kümesine dayalı geliştirilmiştir. MDA, MOF (Meta Object Facility) kullanımını şart koymaktadır. MOF, belirli bir alan diline özel olmayan tüm genel özellikleri içermektedir. Bu özellikler arasında üst-modeller geliştirmek ve bunlarla işlem yapmak için gerekenler bulunmaktadır." [1]

Model dönüşümü işlemi ise belirgin bir modelden diğer bir modele geçiş işlemini tarif etmektedir. MDA tanımında üst-modelleri tanımlamak için tek bir dil (ör, MOF) yer almaktadır. Bu yöntemle farklı üst-modellere dayanan modeller arasındaki uyumluluklar ifade edilebilmektedir. Dönüşümler bu uyumluluğa önemli bir örnektir. Dönüşüm, tek bir dönüşüm dili (üst modelden bağımsız bir dil) ile ifade edilebilmelidir. Dönüşüm dili, alan bağımsız bir dil olmalıdır. UML, yaygın olarak kullanılan bir üst-model sağlamaktadır.

Kod merkezli mimari ve sistem geliştirme yaklaşımında mühendisler kod üzerine yoğunlaşırlar ve sonunda sistemin ana özelliklerini göz ardı edebilirler. Model güdümlü

yaklaşımlarda ise sistemin arkasındaki ana model ele alınır ve daha genel bir perspektife göre sistem geliştirilir.

Bu bildiride öncelikle ontoloji tabanlı model dönüşüm yöntemleri konusunda inceleme yapılacaktır. Bu yöntemlerin incelenmesi sonucunda edinilecek bilgilerden de yararlanarak doktora çalışmam kapsamında UML modellerinden Web Servisi modellerine dönüşüm yapmayı sağlayacak bir araç geliştirilmesi planlanmaktadır.

Bu bildiride kapsamı Model Güdümlü Mimari'ye (MDA) bağlı kalınarak model dönüşümü konusunda yapılan çalışmaları ve ontoloji tabanlı olarak model dönüşümü işleminin gerçekleştirilmesi için gerekli altyapı ve elemanları incelenecektir. Çalışmanın ikinci bölümü model dönüşümleri ve dönüşüm dilleri, üçüncü bölümünde ise Ontoloji Tabanlı Model Dönüşümünün bileşenleri ve dönüşüm adımları açıklanacaktır. Dördüncü bölümde bu konuda literatürde gerçekleştirilen çalışmalar yer alacaktır.

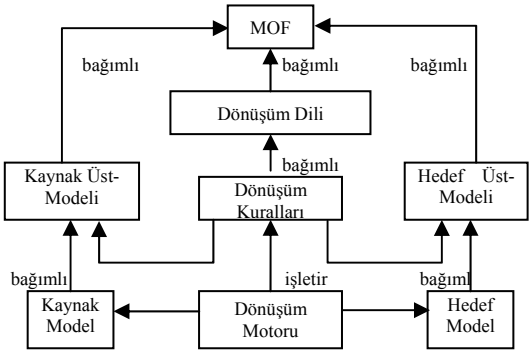
2 Model Dönüşümleri

Model dönüşümü, kaynak bir modelden hedef bir modele, belirli dönüşüm kurallarına bağlı olarak yapılan bir eşleştirme işlemidir. Bu şekilde bir modelin farklı platformlarda kullanılabilecek farklı görünümlerini (modellerini) yaratmak mümkün olabilmektedir.

Model dönüşümü konusunda MDA'deki önemli kavramlar, Platform Bağımsız Model (Platform Independent Model (PIM)), Platform Spesifik Model (Platform Spesific Model (PSM)), model dönüşüm dili, dönüşüm kuralları ve dönüşüm motoru kavramlarıdır. Bu yaklaşımda MOF diğer üst-modellerin yaratılması için kullanılacak üst-üst-modeldir. PIM, sistemin davranışını, yapısını ve fonksiyonlarını yansıtır. PSM, daha çok gerçekleştirime yöneliktir ve bir çalıştırma ortamı için verilen PIM'in belirli bir platform için dönüştürülmüş halidir[3]. PSM, son uygulama olmamakla birlikte, arayüz dosyaları, program kodu, arayüz tanımlama dili, ve konfigürasyon dosyaları gibi kavramların yaratılabilmesi için gerekli bilgiye sahiptir. PIM'den PSM'e eşleme iki üst-model arasındaki eş elemanları kararlaştırır. Farklı iki üst-modeldeki elemanların eş olması demek bunların birbiriyle uyumlu olması ve birbirleriyle çelişmemesi demektir. Model dönüşümü, dönüşüm kurallarını çalıştıran bir dönüşüm motoru tarafından gerçekleştirilir. Dönüşüm kuralları bir kaynak modelden (ör, PIM) bir hedef model (ör, PSM) oluşturulması için gerekli yolları tanımlar. Bir modeli diğer bir modele dönüştürmek için dönüşüm kuralları kaynak üst-modeli, hedef üst-modele eşler. Dönüşüm motoru kaynak modeli alır, dönüşüm kurallarını işletir ve çıkışta hedef modeli verir. Şekil 1'de MDA'de model dönüşüm işlemi görülmektedir.

2.1 Model Dönüşüm Dilleri

Ontoloji tabanlı olarak gerçekleştirilmeyen klasik model dönüşüm işlemi için bir model dönüşüm dili kullanımı gerekmektedir. Bu kapsamda literatürde birçok farklı dil kullanılmaktadır. Bu dillerden en yaygın olarak kullanılanları ise, Atlas Transformation Language (ATL), Graph Rewriting and Transformation Language (GreAT), MOdel transformation LAguage (MOLA) ve Yet Another Transformation Language (YATL) dilleridir. Bu diller hakkında aşağıda kısa bilgiler verilecektir. Bu çalışmada ise bu tip model dönüşüm dili kullanılmadan ontoloji dillerine bağlı olarak model dönüşümünün gerçekleştirilmesi planlanmaktadır.



Şekil 1. MDA'de Dönüşüm [3]

Atlas Transformation Language (ATL). ATL (Atlas Transformation Languge) ATLAS INRIA & LINA araştırma grubu tarafından OMG'nin MOF[15]/QVT[16] RFP kavramlarına cevap olarak geliştirilmiştir [13][14]. ATL, MDA uyumlu bir dildir ve dönüşüm kuralları kullanılarak eşleştirilen kaynak ve hedef üst-modelleri işlemek ve saklamak için bir havuz kullanmaktadır. Hem bir üst model hem de metinsel yazıma sahip bir dönüşüm dilidir. ATL dili tanımlayıcı (*declerative*) ve *imperative* programlama özelliklerini taşır. Bir dönüşümün tanımlanmasında daha çok tanımlayıcı yaklaşım tercih edilmektedir. Bunun sebebi hedef ve kaynak model elemanları arasındaki eşleşmelerin kolayca ifade edilebilmesidir. Bununla birlikte ATL, tanımlayıcı olarak ifade edilmesi zor olan eşleşmelerin tanımlanmasını kolaylaştırmak amacıyla imperatif yapılar da sunmaktadır [13]. Bir ATL dönüşüm programı, hedef model elemanlarının yaratılması ve başlatılmasının kaynak model elemanları ile nasıl yönetileceğini ve bunların eşleştirilmesini sağlayacak kurallardan oluşmaktadır. Temel model dönüşümlerinin yanında, ATL modeller üzerinde istekler tanımlamayı mümkün kılan ek bir model sorgulama servisi de tanımlamıştır [13][14]. Sorgulama OCL ifadeleri kullanılarak yapılmaktadır [13].

Graph Rewriting and Transformation Language (GreAT). GReAT dili, (Graph Rewriting and Transformation Language)[17] grafik yeniden yazımını kullanan üst model tabanlı bir dönüşüm dilidir ve farklı alanlara da ait olabilecek çoklu grafiklerin dönüşümlerini desteklemektedir. Alana Özel Model Yönelimli Mimarilerin (DSMDA (Domain Specific MDA)) gelişimini hızlandırmak için tasarlanmıştır. Alana özel PIM'lerden, alan bağımsız PSM'lere dönüşüm için geliştirilmiştir. GReAT dili üç parçaya bölünmüştür. Bunlar; Desen Tanımlama Dili (Pattern Specification Language), Grafik Dönüşüm Dili (Graph Transformation Language) ve Kontrol Akış Dili'dir (Control Flow Language). Bu dilin en önemli kısımları Desen Tanımlama Dili ve Desen Eşlemedir.

Model Transformation Language (MOLA). MOLA (MOdel transformation LAguage) [18] dilinin temel hedefi grafik formdaki geleneksel yapısal programlamayı

(yapısal akış çizelgeleri şeklinde) desen tabanlı kurallarla birleştirerek kolay okunabilir bir grafiksel dönüşüm dili ortaya koymaktır [18]. MOLA'da içsel olarak özyinelemeliden (recursive) çok yinelemeli (iterative) olan tipik model dönüşüm algoritmalarının çoğu doğal bir şekilde tanımlanabilmektedir [18]. Bir MOLA programı kaynak üst modele uyumlu bir kaynak modelin, hedef üst modele uyumlu olarak hedef modele dönüşümü için kullanılmaktadır. Her iki model de uygun üst modeli karşılayan sınıf ve ilişki olay kümeleri olarak ele alınmaktadır [18].

Yet Another Transformation Language (YATL). YATL Dili (Yet Another Transformation Language) [19], Kent Modelling Framework [20] kapsamında geliştirilmiş bir dönüşüm dilidir. Derleyicisi ve yorumlayıcısı Java ile geliştirilmiştir ve farklı modelleme ortamları ve araçlarının taşınabilirliğini maksimize amacıyla tasarlanmıştır. Bir YATL dönüşümü tek yönlüdür. Dilin geliştiricileri büyük ölçekteki modellerde de dilin kullanılabilmesi için model dönüşüm dilinin tek yönlü olması gerektiğini savunmaktadır [19]. Buna sebep olarak da çift yönlü bir dönüşüm dilinin boşlukları doldurmak için birleştirme operasyonu kullanan bir çıkarsama makinesi gibi çalışacağını göstermektedirler [19].

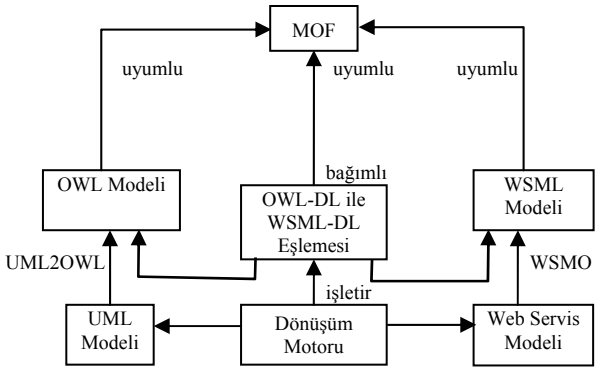
3 Ontoloji Tabanlı Model Dönüşümü

Sözlük anlamı "varlık bilimi" olarak tanımlanan ontolojilerin bilgisayar biliminde en çok kabul gören tanımı ise "kavramsallaştırmanın biçimsel ve açıkça belirtilmesi" olarak ele alınır. [21] Burada kavramsallaştırma ile kast edilen, belirli bir ön alanda, bu ön alana ait soyut model oluşturma anlamına gelmektedir. Ontolojiler herhangi bir alanda standart olarak kullanılacak ortak ve paylaşılan sözcük kümelerini (vocabulary) veya terminolojiyi belirler. Anlamsal olarak bütünlüğün ve uygunluğun sağlanmasına yardımcı olurlar. Ontolojiler, Anlamsal Web'de bilginin anlamlı paylaşılabilmesi için kullanılmaktadırlar. Ontolojiler sayesinde sınıflandırmanın ötesinde kavramlar arasında anlamsal ve mantıksal bağlantılar kurabilmek mümkün olmaktadır.

Ontolojiler kullanılmadan yapılan model dönüşümlerinde, dönüşüm işlemi daha çok sözdizimsel ve yapısal olarak gerçekleştirilmektedir. Modellerin içinde barındığı anlamın korunması konusunda kontroller genelde yapılmamaktadır ve anlamın korunması garanti edilememektedir. Model dönüşümlerinde ontoloji kullanılmasının getirebileceği yararlar anlamsal (semantik) olarak birbiri ile tam eşleşen yapıların ortaya çıkmasıdır. Bunun sebebi ise bir alandaki kavramların diğer bir alandaki kavramlara ontolojiler kullanılarak dönüşümü (eşlenmesi) durumunda anlamsallığı korumak adına eşlemelerin çeşitli kurallara bağlı kalınarak gerçekleştirilme zorunluluğudur. Ontoloji kavramının doğası itibariyle model dönüşümünün ontolojiler kullanılarak gerçekleştirilmesi yoluyla, anlamsal olarak birbirine uyumlu modeller oluşturulabilecektir. Ontoloji kullanılarak yapılan model dönüşümünde, modeller ontoloji seviyesine çıkarılmaktadır ve üst-modeller, ontolojiler ve ontoloji dilleri kullanarak gerçekleştirilmektedir. Model dönüşümü işlemi ise ontolojilerin birbiriyle eşlenmesi yoluyla gerçekleştirilmektedir. Anlamsal bütünlüğün korunması; belirli bir servisin aranması sırasında daha uygun ve gereksinimleri karşılayan servislerin bulunmasını sağlamaktadır. Ayrıca ontoloji kullanımı ile model dönüşümü işlemi daha fazla makine tarafından anlaşılır (machine understandable) bir formda

gerçekleştirilebilir. Ontoloji tabanlı model dönüşümü yeni bir araştırma konusudur ve bu konuda yapılan çalışmalar oldukça az sayıdadır.

Bu bölümde model dönüşümü işleminin ontoloji olarak gerçekleştirilme yolları, bileşenleri ve dönüşüm işleminde yer alması gereken adımlar yer almaktadır. Bu kapsamda model dönüşüm işleminde, kaynak model olarak UML modeli, hedef modeli olarak ise Web Servis Modeli ele alınacaktır. MDA tabanlı olarak böyle bir dönüşüm işleminin girdi ve çıktıları Şekil 2'de görülmektedir. Aşağıdaki şekil; MDA'deki temel model dönüşümü işleminin [3] bu çalışmada ontolojiler kullanılarak yapılması planlanan model dönüşümü işlemi için uyarlanmış şeklidir. MDA'deki PIM bu çalışmada UML modeli, PSM ise Web Servis Modeli olarak ele alınmıştır. Üst modeller ise OWL Modeli ve WSML Modeli olarak ele alınacaktır.



Şekil 2. Model Dönüşüm Basamakları

Ontoloji tabanlı model dönüşümü işlemi için iki farklı model dönüşümünün birbiri ile entegre edilmesi gerekmektedir. Birincisi UML modelinden OWL modeline olan, yani model katmanından üst-model (ontoloji) katmanına olan UML2OWL dikey dönüşümü, ikincisi ise OWL ontoloji modeli ile WSML ontoloji modeli arasında gerçekleştirilmesi gereken yatay model dönüşümü işlemidir. Çalışmada yapılması planlanan bu model dönüşümlerini entegre edecek bir dönüşüm motorunun geliştirilmesi yoluyla ontoloji tabanlı model dönüşümünün gerçekleştirilmesidir.

Aşağıda, bu dönüşüm işleminin bileşenleri olan Web Servis Modelleme Ontolojisi (Web Service Modelling Ontology (WSMO)) ve Web Servis Modelleme Dili (Web Service Modelling Language (WSML)) ile UML'den OWL diline dikey model dönüşümü için kullanılacak UML2OWL dönüşümü ve hakkında bilgi verilecektir. Ayrıca OWL ve WSML ontolojileri arasındaki yatay model dönüşümü de açıklanacaktır. Bu teknolojilerin kullanımı ile kaynak ve hedef modeller, ontoloji katmanına çıkartılabilmekte ve model dönüşümünün ontolojiler kullanılarak gerçekleştirilebilmesi için bir ortam hazırlanmaktadır.

3.1 Ontoloji Tabanlı Model Dönüşüm Bileşenleri

Web Servis Modelleme Ontolojisi ve Dili (WSMO ve WSML). Digital Enterprise Research Institute (DERI) tarafından geliştirilen WSMO [8], web servislerinin arama, birleşme ve çağırma gibi elektronik servislerinin web üzerindeki otomasyonunu anlamsal olarak tanımlayabilecek bir dil ve kavramsal bir çerçeve sağlamaktadır.

WSMO'nun dört temel elemanı vardır:[9]

- ***Ontolojiler***: WSMO elemanları tarafından kullanılan terminolojiyi sağlar.
- ***Web Servis Tanımları***: Web servislerin fonksiyonel ve davranışsal özelliklerini tanımlar.
- ***Hedefler***: Kullanıcıların isteklerini ortaya koyar.
- ***Arabulucular (Mediator)***: Farklı WSMO elemanları arasındaki birlikte çalışırlık problemleri ile ilgilenir.

WSMO'da kullanılan mantıksal dile Web Servis Modelleme Dili (WSML) adı verilmektedir. WSMO, anlamsal web servisleri ile ilgili kavramlar için bir üst-modeldir. Bu modeli tanımlamak için MOF spesifikasyonu kullanılmaktadır. Teknoloji olarak nötr üst-modelleri tanımlamak, inşa etmek ve yönetmek için gerekli dil ve çerçeve MOF tarafından tanımlanmaktadır.

MOF dört katmandan oluşan bir üst-veri mimarisi tanımlar:[9]

- Bilgi katmanı, tanımlanacak veriyi ortaya koyar.
- Model katmanı, bilgi katmanındaki veriyi tanımlayan üst-veriyi içerir
- Üst-model katmanı, üst-verinin yapı ve anlamlarını tanımlayan tanımlamaları içerir.
- Üst-üst-model katmanı ise üst-üst-verinin yapı ve anlamını içerir.

MOF katmanları ile karşılaştırıldığında WSMO'yu tanımlayan dil üst-üst-model katmanına denk gelir. WSMO'nun kendisi üst-model katmanını oluşturur. Ontolojiler, web servisleri, hedefler ve arabulucuların spesifikasyonları ise model katmanını oluşturur. Ontolojiler tarafından tanımlanan ve web servisleri tarafından alınıp verilen veri ise bilgi katmanını oluşturur. [9]

Web Servis Modelleme Dili olan WSML ise web servislerinin WSMO'ya bağlı olarak modellenmesi amacıyla kullanılmaktadır. WSML'in, ontoloji tabanlı model dönüşümü çalışmasındaki yeri ise PSM model olarak kullanılacak web servisi modelinin ontoloji katmanına çıkarılması (Şekil 2) aşamasında kullanılacak bir üst-model dili olmasıdır. Aşağıda örnek bir WSML kodu yapısı görülmektedir (Şekil 3) :

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-dl"
namespace { _"http://www.example.org/ontologies/example#",
  dc  _"http://purl.org/dc/elements/1.1#"}
ontology _"http://www.example.org/ontologies/example"
concept Animal
```

```
concept Human
  hasFather impliesType Man
  hasWeight ofType {_decimal, _integer}
  hasBirthday ofType _date
concept Person subConceptOf Human
concept Woman subConceptOf{Human}
  nfp dc#relation hasValue "disjoint to Man" endnfp
concept Child subConceptOf{Human}
  hasParent impliesType Human
relation ageOfHuman (impliesType Human, ofType _integer)
  nfp
    dc#description hasValue "describes the age of a person"
    dc#hasRights hasValue _http://www.deri.org/privacy.htm"
  endnfp
```

Şekil 3. WSML Kod Örneği

UML Modelinden OWL Modeline Dönüşüm (UML2OWL). UML2OWL, bir ODM spesifikasyonuna (QVT eşleştirmesi) bağlı kalınarak bir ATL dili (Atlas Transformation Language) model dönüşümü olarak gerçekleştirilmiştir.[8] Bu dönüşüm, bir UML modelinin OWL ontolojisine dönüştürülmesini mümkün kılmaktadır. Bu çalışmadaki yeri ise PIM olarak kullanılacak olan UML modelinden ontoloji dili olan OWL diline dönüşüm olarak kullanılmasıdır (Şekil 2).

UML modelleri, bir seri üst-seviye (meta-level) şeklinde düzenlenmiştir: Bunlar, M3, M2, M1 ve M0'dır.[8]

- M3, modelleme sistemlerinin tanımlandığı üst-üst-model modelleme dili olan MOF'tur.
- M2, seçilen modelleme sisteminin modelidir. UML üst-modeli bir M2 yapısıdır ve M3'te MOF olarak tanımlanmıştır.
- M1, belirli bir modelleme sisteminde temsil edilen belirli bir uygulamanın modelidir.
- M0, ise belirli bir uygulamadır.

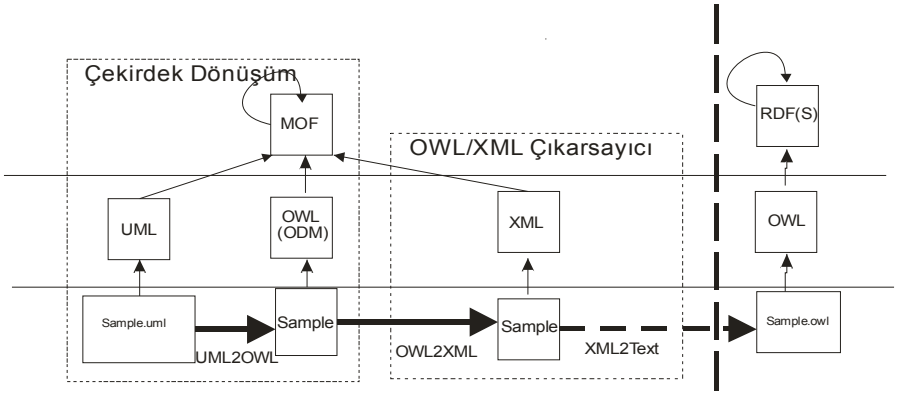
OMG'nin ODM Spesifikasyonu'nda, UML modeli, OWL modellerine eşleştirilmiştir. Eşleştirmeler, OWL-DL ile sınırlandırılmıştır. Bu eşleştirme tanımlamalarında yalnızca OWL-DL yapılarının kullanılacağı anlamına gelmektedir. UML2 kernel paketinde birçok "abstract" üst-sınıflar mevcuttur. Böylece sadece önemli sınıflar OWL yapıları ile eşlenmiştir. Eşlemeler QVT (MOF QVT) ile ifade edilmiştir. [8]

Şekil 4'te ise UML2OWL dönüşüm senaryosu gösterilmektedir. Bu senaryo iki ATL dili dönüşümünün bileşiminden oluşmaktadır:[8]

- Ana dönüşüm olan UML2OWL, girdi olarak bir UML modeli alır ve OWL üst-modeline uygun bir ontoloji üretir.

- İkinci dönüşüm ise, OWL/XML sözdizimine uygun bir XML dokümanı üreten W3C spesifikasyonu tarafından tanımlanmış bir XML çıkarsayıcıdır (extractor).

Ana dönüşümün iki ayrı parçası mevcuttur.[8] Birinci kısım, UML modelinden ontolojiye eşleştirmedir. Örneğin, UML sınıfları OWL sınıflarına, özellikler (attribute) veri tipi özelliklerine (datatype property), ilişkiler (association) ise nesne özelliklerine (object property) eşlenir. İkinci kısım ise örneklerle (instance) ilgilidir.



Şekil 4. UML2OWL Dönüşüm Senaryosu [8]

Basitlik ve gösterim kolaylığı bakımından örnekler, UML modeli ile aynı sınıf diyagramında tanımlanmıştır. Bu örnekler OWL tekillerine (individual - OWL dilindeki örneğin karşılığı) çevrilmiştir. Bu metod, UML örneklerini yönetme ve ilgili bilgilerle ontoloji oluşturmayı mümkün kılmaktadır. [8]

UML2OWL dönüşümü "ecore" formatında bir OWL modeli veya OWL/XML sözdizimine uygun bir OWL dokümanı üretebilir. Bu XML dosyasını elde etmek için OWL üst-modelini bir OWL dokümanına çeviren bir OWL/XML çıkarsayıcısı geliştirilmiştir. Bu çıkarsayıcı, geliştirilen OWL dosyalarının Protege gibi ontoloji araçlarında kullanılmasını sağlamaktadır.

3.2 Yatay Model Dönüşümü Adımları

Ontoloji kullanarak yapılan model dönüşüm işleminde öncelikle ontoloji eşleştirmesi konusunu incelemek gerekmektedir. Bu çalışma kapsamında ise OWL ile WSML dilleri arasındaki yatay model dönüşümü için gerekli eşleştirmeye bakılmalıdır.

OWL-DL İle WSML-DL Eşleştirmesi. Bu bölümde OWL-DL'den WSML-DL'e eşleme incelenecektir. Bu eşleme çift taraflı olarak yapılabilmektedir. Bu eşlemenin çalışmadaki önemi ise Şekil 2'deki PIM ve PSM üst-modelleri olan OWL ve WSML

arasındaki yatay model dönüşümün gerçekleştirilmesi konusunda olacaktır. Bu çalışma kapsamında OWL ontolojileri ile WSML ontolojileri arasındaki yatay model dönüşüm işlemi için önemli bir adım oluşturmaktadır. Eşleme WSML-Core'dan OWL-DL'e dayanır. OWL-DL'e eşleme WSML ontolojileri ve mantıksal ifadelere uygulanabilir. [12]

Önişleme Adımları. WSML-DL'den OWL-DL'e dönüşümü basitleştirmek için bir takım önişleme adımları uygulanmıştır. Bunlardan bazıları aşağıda görülmektedir: [12]

- Bütün veri terimleri kısa yolları değiştirilir.
  - ("string":=string("string");integer:=integer(integer); decimal:=decimal(decimal)).
- Bütün WSML veri tipi üreticileri denk gelen XML Şema veri tiplerine göre değiştirilir.
  - string → " http://www.w3.org/2001/XMLSchema#string" ile değiştirilir.

Eşleştirme Tablosu. Aşağıdaki Tablo 1'de [12], OWL-DL *"abstract"* sözdizimi ile WSML-DL sözdizimi arasındaki eşleştirme yer almaktadır. Eşleştirme işlemi, eşleştirme fonksiyonu "τ" tarafından tanımlanır.

Tablo 1. WSML-DL ile OWL-DL Arasındaki Eşleştirme [12]

WSML-DL	OWL-DL	Yorumlar
Ontolojiler için eşlemeler		
τ (ontology id başlık$_1$ … başlık$_m$ ontoloji_elemanı$_1$ … Ontoloji_elemanı$_m$)	Ontology(id τ(başlık$_1$) … τ(başlık$_m$) τ(ontoloji_elemanı$_1$) … τ(ontoloji_elemanı$_m$))	Bir başlık sadece *"nonFunctionalProperties", "usesMediator"* ve *"importsOntology"* cümlelerini içerebilir. Bir ontoloji elemanı bir kavram, ilişki, örnek, ilişki örneği veya bir aksiyom olabilir.
τ(nonFunctionalProperties id$_1$ hasValue değer$_1$ .... İd$_n$ hasValue değer$_n$ endNonFunctionalProperties)	Annotation(id$_1$ τ(değer$_1$)) …. Annotation(id$_n$ τ(değer$_n$))	Fonksiyonel olmayan özellikler için ontoloji seviyesinde "annotation" yerine "Annotation" yazılmalıdır.
τ(importsOntology id)	Annotation(owl#import id)	"id" WSML dosyasının tanımlayıcısı yerine geçer.
τ(datatype_id(x$_1$,…,x$_n$))	datatype_id(x$_1$,…,x$_n$)^^ T$_{datatypes}$ (datatype_id)	T$_{datatypes}$, WSML veritiplerini [9]'a göre XML veritiplerine eşler

Şekil 3'teki kodun WSML2OWL dönüşümü aracılığıyla OWL diline dönüştürülmüş hali Şekil 5'te görülmektedir.

```
Namespace(rdf  = <http://www.w3.org/1999/02/22-rdf-syntax-
ns#>)
Namespace(xsd  = <http://www.w3.org/2001/XMLSchema#>)
Namespace(rdfs = <http://www.w3.org/2000/01/rdf-schema#>)
Namespace(owl  = <http://www.w3.org/2002/07/owl#>)
Namespace(a    =
<http://www.example.org/ontologies/example#>)

Ontology( <http://www.example.org/ontologies/example>

ObjectProperty(a:hasFather domain(a:Human) range(a:Man))
ObjectProperty(a:hasParent domain(a:Child)range(a:Human))
DatatypeProperty(a:ageOfHuman domain(a:Human)
range(xsd:integer))
DatatypeProperty(a:hasBirthday domain(a:Human)
range(xsd:date))
DatatypeProperty(a:hasWeight domain(a:Human)
range(xsd:decimal) range(xsd:integer))
Class(a:Child partial a:Human)
Class(a:Human partial)
Class(a:Man partial)
Class(a:Person partial a:Human)
Class(a:Woman partial a:Human)
SubClassOf(a:Human restriction(a:hasFather
allValuesFrom(a:Man)))
SubClassOf(a:Human restriction(a:hasBirthday
allValuesFrom(xsd:date)))
SubClassOf(a:Human restriction(a:ageOfHuman
allValuesFrom(xsd:integer)))
SubClassOf(a:Human restriction(a:hasWeight
allValuesFrom(xsd:decimal)))
SubClassOf(a:Child restriction(a:hasParent
allValuesFrom(a:Human)))
SubClassOf(a:Human restriction(a:hasWeight
allValuesFrom(xsd:integer)))
}
```

Şekil 5. WSML2OWL Dönüşümü Sonucu Ortaya Çıkan OWL Kodu

4 Literatürdeki Çalışmalar

Bu bölümde ontoloji tabanlı model dönüşümü konusunda yapılmış olan bazı çalışmalardan kısaca bahsedilecektir.

Ontoloji tabanlı model transferi konusunda DERI (Digital Enterprise Research Institute) tarafından birçok araştırma ve çalışma yürütülmektedir. Bu kapsamda yukarıda

belirtilen WSML dilinden OWL diline eşleme ve dönüşüm için bir web servisi geliştirilmiştir. Ayrıca WSMO ve WSML dili ile beraber bu teknolojiler ile beraber çalışacak Web Service Modelling Toolkit (WSMT), Web Service Execution Environment (WSMX) gibi birçok ortam bu çalışma grubu tarafından geliştirilmiştir. [9][10][11]

Ayrıca Stephan Roser ve Bernhard Bauer tarafından da ontoloji tabanlı model dönüşümleri konusunda araştırmalar yürütülmektedir. Bu çalışmada ise "Semantic Enabled Model Transformation Tool (sem-MT-Tool)" adını verdikleri bir araç geliştirmişlerdir. [22][23] Bu araç "ontMT" adını verdikleri bir kavramı gerçekleştirerek model dönüşümü işlemini otomatikleştirmeyi amaçlamaktadır. Bu araçla bir referans ontolojisine ve çıkarsama teknolojilerini kullanarak otomatikleştirilmiş ontoloji tabanlı model dönüşümü gerçekleştirmeyi amaçlamaktadırlar. Bu araştırmada MDA teknolojik alanı ile Ontoloji teknolojik alanı arasında otomatikleştirilmiş dönüşümler gerçekleştirilmektedir.

[22] ve [23]'teki araştırma ile gerçekleştirilmesi planlanan çalışma arasında benzer yönler bulunmasına rağmen; çalışmanın MDA tabanlı bir model dönüşümünü ontolojiler kullanılarak gerçekleştirmeyi planlaması dolayısıyla farklılıklar göstermektedir. Diğer çalışmada ise MDA ve Ontoloji teknolojik alanları (Technological Space) arasında otomatikleştirilmiş model dönüşümleri gerçekleştirilmektedir.

5 Sonuç

Bu bildiride, model güdümlü mimari, model dönüşümleri, model dönüşüm dilleri ve ontoloji tabanlı model dönüşümü konusunda literatür çalışması yapılmıştır. Bununla birlikte ontoloji tabanlı model dönüşüm işlemi için gerekli bileşenler açıklanmış ve bu bileşenlerin birlikte kullanımı ile model dönüşümünün nasıl yapılabileceği ile ilgili fikirler ortaya konmuştur.

Yapılan araştırmalar neticesinde doktora çalışmamda gerçekleştirilmesi planlanan ontoloji tabanlı model dönüşümü işlemi için kaynak modelin, OWL seviyesine, hedef modelin ise WSML seviyesine çıkarılabilme yolları bulunmuştur. Ayrıca OWL ile WSML eşleştirmesinin nasıl yapılabileceği de belirlenmiştir. Dolayısıyla model dönüşümü işleminin tüm bileşenleri ortaya çıkarılabilmiştir.

Sonuç olarak ontoloji tabanlı model dönüşümü işlemi ile anlamsal bütünlüğün korunmasının sağlanabileceği ve birbirleriye tam uyumlu modeller yaratılabileceği düşünülmektedir.

Doktora çalışması kapsamında gelecekteki çalışmalarda bir dönüşüm motoru geliştirilmesi yoluyla mevcut bileşenlerin birlikte çalışması sağlanabileceği ve bu yol ile ontoloji tabanlı model dönüşümü işleminin gerçekleştirilmesi düşünülmektedir.

Kaynaklar

1. Bezivin J., Gerard S., "A Preliminary Identification of MDA Components", in Generative Techniques in the context of Model Driven Architecture, Kasım 2002.

2. Dustdar S., ve Schreiner W., “A Survey on Web Services Composition”, Int. J. Web and Grid Services, Vol. 1, No. 1, 2005.

3. Bezivin J., vd., “Applying MDA Approach for Web Services Platform”, Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conference (EDOC 2004).

4. OWL Services Coalition; “OWL-S 1.0: Semantic Markup for Web Services”; http://www.daml.org/services/, 2003.

5. SOAP, http://www.w3.org/TR/soap, 2003.

6. WSDL 2.0, http://www.w3.org/TR/wsdl20, 2006.

7. Bezivin J., “From Object Composition to Model Transformation with the MDA”, TOOLS USA, Volume IEEE TOOLS-39, Santa Barbara, Ağustos 2001.

8. “Ontology Definition Metamodel Specification”, OMG Adopted Specification, ptc/06-10-11

9. Bruijn J., vd., “Web Service Modeling Ontology (WSMO)”, W3C Member Submission, Haziran 3, 2005.

10. Bussler C., vd., “Web Service Execution Environment (WSMX)”, W3C Member Submission 3 Haziran 2005

11. Haller A., vd., “WSMX - A Semantic Service-Oriented Architecture”, Digital Enterprise Research Institute (DERI), 2005.

12. Keller U., vd., “RW2 Project Deliverable, D1.2 v1.0, Report on reasoning techniques and prototype implementation for the WSML-Core and WSMO-DL languages”. Document Version from Temmuz 1, 2006.

13. ATLAS Group, “ATL: Atlas Transformation Language ATL User Manual v0.7”, LINA&INRIA Nantes, Şubat 2006.

14. ATLAS Group, “ATL: Atlas Transformation Language,ATL Starter’s Guide, version 0.1”, LINA&INRIA Nantes, Aralık 2005.

15. OMG/MOF Meta Object Facility (MOF) 1.4. Final Adopted Specification Document. formal/02-04-03, 2002.

16. OMG/RFP/QVT MOF 2.0 Query/Views/Transformations RFP. Ekim 2002.

17. A. Agrawal; “GReAT: A Metamodel Based Model Transformation Language”, ASE 2003

18. A. Kalnins, J. Barzdins, E. Celms; “Basics of Model Transformation Language MOLA”, University of Latvia, IMCS, 2004.

19. O. Patrascoiu; “YATL: Yet Another Transformation Language”, Computing Laboratory, University of Kent, United Kingdom.

20. S. Kent, O. Patrascoiu; “Kent Modelling Framework Version – Tutorial”, University of Kent, Canterbury, UK, Draft, 6 Aralık 2002.

21. Orhan Hançerlioğlu, Felsefe Sözlüğü, Remzi Kitabevi, İstanbul, 1994, s. 439.

22. Roser S., Bauer B., “An Approach to Automatically Generated Model Transformations Using Ontology Engineering Space”, In Proceedings of Workshop on Semantic Web Enabled Software Engineering (SWESE), Athens, GA, U.S.A., 2006.

23. Roser S., Bauer B., “Ontology Based Model Transformation”, LNCS Volume 3844/2006.

MİMARİ MODELLEME – TASARIM

HLA Federasyon Mimarileri İçin Federe Arayüz Spesifikasyon Kütüphanesi

Tansel Dökeroğlu[1], Okan Topçu[2], Halit Oğuztüzün[3]

[1,2,3] ODTÜ, Bilgisayar Mühendisliği Bölümü, 06531, İnönü Blv., Ankara
[1] tansel.dokeroglu@msb.gov.tr, [2] okantopcu@gmail.com,
[3] oguztuzn@ceng.metu.edu.tr

Özet. Yüksek Seviye Mimarisi *(HLA),* özerk simülasyonların birlikte çalışabilirliği için yayınla/abone ol paradigmasına dayalı ortak bir altyapı sunmaktadır. *HLA* uyumlu federasyon mimarilerinin betimlenmesine olanak sağlayan ve otomasyonu destekleyen bir metamodel olan Federasyon Mimari Metamodeli *(FAMM),* aynı zamanda muhtelif *HLA* federe arayüz spesifikasyon kütüphanelerinin tanımlanmasına da olanak sağlamaktadır. *HLA* federe arayüz spesifikasyon kütüphaneleri, bir federenin gözlenebilir davranışlarının modellenmesi ve ardından otomatik kod üretimi için kullanılmaktadır. Metamodel, güncel standart olan *IEEE HLA 1516* Metod kütüphanesini içermektedir. Pratik ihtiyaçlardan dolayı *DMSO HLA 1.3* arayüzlerinin desteklenmesi gerektiğinde, metamodelin yapısına dokunmadan, sadece yeni bir kütüphane eklenerek sorunun çözümüne ulaşılmıştır. Bu bildiride *FAMM*'ın kütüphane mekanizması aracılığı ile sağladığı esneklik gösterilmekte ve federe arayüz spesifikasyon kütüphanelerinin kullanımı, *DMSO HLA 1.3* metod kütüphanesi örneği ile anlatılmaktadır.

1 Giriş

Yüksek Seviye Mimarisi *(HLA–High Level Architecture)* [1], [2], [3] özerk simülasyonların karşılıklı çalışabilirlik ve tekrar kullanılabilirlik prensipleri doğrultusunda geliştirilmiş, nesne temelli bir haberleşme altyapısı sağlayan ve yayınla/abone ol paradigmasına dayanan bir dağıtık simülasyon mimarisidir. *HLA*, dağıtık ortamlardaki simülasyon ve modellemeler için *US DoD* ve *NATO* başta olmak üzere tercih edilen bir standart haline gelmiştir. Askeri olduğu kadar sivil alanda da HLA uyumlu simülasyon uygulamaları sıkça görülmektedir.

HLA federasyonlarının geliştirilmesi için oldukça çaba harcanmış olmasına rağmen, halen federasyon geliştirme otomasyonu bakımından, tasarım ve dökümantasyon olarak istenen seviyede olunduğu söylenemez [4]. Ancak son dönemde yapılan çalışmalar ile *HLA* uyumlu simülasyon geliştirilmesi konusunda, verilen bir federasyon mimari modelinden, otomatik kod üretiminin mümkün olduğunu söyleyebiliriz [5]. Simülasyon kavramsal modelinden federasyon mimari modeline geçiş çalışmaları ise halen devam etmektedir[6].

Otomatik kod üretimi için kaynak model, kavramsal model dönüşümü için ise hedef model olarak *Federasyon Mimari Modeli*[1] kritik bir noktada bulunmaktadır. Önceki çalışmalar ile federasyon mimarilerinin geliştirilmesine olanak sağlayan ve otomasyonu destekleyen bir Federasyon Mimari Metamodeli *(FAMM)* önerilmiştir [7]. Önerilen bu metamodel, Uygulama Alanına Özel Metamodelleme yaklaşımının *HLA* uyumlu federasyonlara adapte edilmesiyle federasyon için biçimsel bir gösterim ve uygulama alanına yönelik bir dil sağlamaktadır. Bu metamodel, federasyonun yapısal ve dinamik özelliklerini aynı ölçüde dikkate almaktadır.

FAMM, HLA uyumlu bir federasyon mimarisinin tanımlanabilmesine olanak sağlayan bir metamodeldir. Bu metamodel tanımlandığı etkinlik alanına özgü bir dil olarak algılanabilir. Böylece, tanımlanan dile uygun olarak modeller üretilmesine olanak sağlanmış olur. *FAMM,* dönüşümlerin tanımlanmasını hem kaynak hem de hedef model bakımından desteklemektedir. Özellikle, simülasyonun kavramsal modelinden dönüşüm yapılmasını ve tanımlanmış federe davranışlarından federe iletişim kodunun üretilmesini desteklemektedir.

*FAMM'*ın öne çıkan özelliği Canlı Sıralama Çizelgelerine *(LSC–Live Sequence Charts)* dayalı olarak federelerin davranışlarının tanımlanabilmesine olanak vermesidir. *FAMM,* federeler için *HLA* Arayüz Tanımlamasını ve *HLA* Standart Nesne Modelini formalize eder. *FAMM,* özellikle kod üretimi için otomatik araçlar tarafından işlenmeyi destekler. Jenerik Modelleme Ortamının *(GME – Generic Modeling Environment)* metamodeli olan *MetaGME* kullanılarak oluşturulmuştur. Jenerik Modelleme Ortamı[2] Vanderbilt Üniversitesi tarafından geliştirilmiş bir etkinlik alanına özel metamodel ve model oluşturma aracıdır. Hem *FAMM*'ın oluşturulmasında hem de bu metamodele dayalı Federasyon Mimarilerin oluşturulmasında aynı araç kullanılabilmektedir [8]. *FAMM* ile ilgili ayrıntılı bilgiye [4 ve 7]'den ulaşılabilir.

FAMM, federasyon mimarilerinin bir parçası olarak davranış modeli tanımlanmasını desteklediğinden, ihtiyaç duyulan *HLA* federe arayüz servislerinin de önceden modellenmesine olanak sağlar. *HLA* federe arayüz servisleri bir *GME* kütüphanesi olarak bir defaya mahsus olmak üzere hazırlanır ve federasyon geliştiricileri tarafından federasyon mimarilerinin modellenmesinde, özellikle bir federenin davranışlarının modellenmesinde, hazır kütüphaneler olarak kullanılır.

FAMM ile birlikte *IEEE HLA 1516* ve *DMSO HLA 1.3* federe arayüz spesifikasyon kütüphaneleri[3] sunulmaktadır [4].

Bu bildiride *DMSO HLA 1.3* Federe Arayüz kütüphanesinin tanıtılması ve geliştirilme sürecinin açıklanması hedeflenmiştir. Ayrıca *FAMM*'ın sağladığı genişletme desteği tartışılacak ve federe arayüz spesifikasyon kütüphanelerinin kullanımı bir örnek ile anlatılacaktır.

---

[1] Bildiri boyunca Federasyon Mimari Modeli ve Federasyon Mimarisi eş anlamlı olarak kullanılmaktadır. Federasyon kelimesi ile *HLA* uyumlu federasyon kastedilmektedir.
[2] Bu çalışmada *GME* versiyon 7.6.29 kullanılmıştır.
[3] Kısaca servis kütüphanesi/kütüphaneleri terimi kullanılacaktır.

FAMM kullanılarak, IEEE HLA 1516 standardından daha eski olan *DMSO* arayüz spesifikasyonunun desteklenmek istenmesinin nedeni, üniversitemizdeki öğrencilerin ders projeleri için *DMSO RTI NG 1.3* üzerinde çalışacak federe kodu üretebilmelerini sağlamaktır.

İlerideki bölümlerde öncelikle *HLA* Federe Arayüz Spesifikasyon Kütüphanesi kavramı açıklanacak ve *DMSO HLA 1.3* Kütüphanesi tanıtılacaktır. Üçüncü bölümde Kütüphane oluşturma çalışmaları anlatılacak, Dördüncü bölümde ise kullanıcı perspektifinden örnek bir uygulama sunulacaktır. Sonuç bölümü ile konu anlatımı tamamlanacaktır.

2 FAMM İçin HLA Federe Arayüz Spesifikasyon Kütüphaneleri

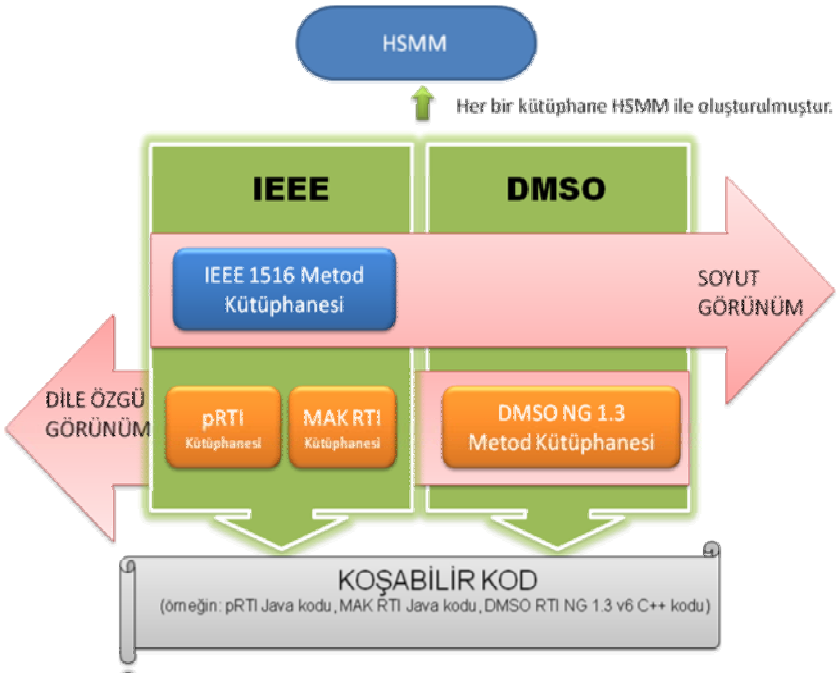
HLA standartları federe arayüzünü açık bir şekilde tanımlamaktadır. *HLA* federe arayüz servisleri olarak adlandırılan bu servisler federasyon yönetimi, tanımlama yönetimi, nesne yönetimi, sahiplik yönetimi, zaman yönetimi, veri dağıtım yönetimi ve destek servisleri başlıkları altında toplanmaktadır [2].

HLA arayüz servisleri, kullanılan standarda göre farklılıklar göstermektedirler. *HLA* mimarisini tanımlayan *IEEE* standartları olmasına rağmen, *HLA*'in gelişim aşamasında *DMSO* tarafından belirlenen *DMSO* federe arayüz servisleri de halen geniş bir şekilde kullanılmaktadır. Bunlara ilave olarak *HLA* Koşum Altyapısı *(RTI)* geliştiricileri bu arayüz servisleri için birden fazla metod da tanımlayabilmektedirler. Örneğin, *Pitch pRTI, attributeIsNotOwned* ve *AttributeIsOwnedByRTI* gibi metodlar tanımlanmaktadır. Bu tip eklentilerin ötesinde, projelere özel olan ve federe arayüz servislerini sarmalayan ara katmanlar da kullanılmaktadır. Ayrıca, *HLA* standartları zaman içinde gelişim göstermektedir (*HLA Evolved* gibi). Bu gelişimlerde olabilecek değişimleri öngörmek mümkün değildir. Görüldüğü gibi *HLA* federe arayüz spesifikasyonu hem geniş bir yelpazeye yayılmakta, hem de simülasyonlar tarafından değişik *HLA* federe arayüz spesifikasyonları (*HLA 1.3* ve *HLA 1516* standartlarının üreticiye özel yorumlamaları gibi) kullanılmaktadır.

HLA federe arayüz spesifikasyonlarını dinamik olarak mümkün olduğunca geniş bir yelpazede destekleyebilmek ve öngörülemeyen değişiklikler olduğunda en düşük düzeyde etkilenmesini sağlamak için, *FAMM*'ın geliştirilmesinde *HLA* federe arayüz spesifikasyonunda belirlenen servislerin metamodelde tanımlanması yerine, yalnızca bu servisleri oluşturan temel elemanların (metod, argüman gibi) *FAMM*'da yer almasına karar verilmiştir. Bu sayede mevcut olan tüm arayüz servisleri *FAMM* kullanılarak modellenebilmekte ve *GME* kütüphanesi oluşturulabilmektedir. Aslında *GME* kütüphanesi oluşturulmuş bir modeldir. Oluşturulan bu modeller *GME* projeleri (model oluşturma amacıyla tasarlanan projeler) içinde kullanılabilmektedirler. Kütüphane modelleri, ithal edildikleri modeller içinde değiştirilemez ve oldukları gibi kullanılır. *IEEE HLA 1516* yanısıra bu çalışma ile *DMSO HLA 1.3* Metod Kütüphanesine de FAMM içinde destek sağlanmıştır.

Şekil 1'de görüldüğü gibi *FAMM* kullanılarak geliştirilen kütüphaneler temel olarak *DMSO* ve *IEEE* Standardına dayalı olmak üzere ikiye ayırılırlar. Buna ek olarak servis kütüphaneleri "Programlama Diline Özgü" yada "Soyut (bir programlama diline özgü olmayan)" şeklinde sınıflandırabilir. Örneğin, *DMSO HLA 1.3* Metod Kütüphanesi programlama diline özgü *(C++)* ve *DMSO* standardına dayalı bir servis kütüphanesidir.

Öte yandan *IEEE HLA 1516* Metod Kütüphanesi soyut ve *IEEE* standardına dayalı bir kütüphanedir. İsteyen modelleyiciler *FAMM* kullanarak RTI'a özgü (*pRTI, MAK RTI* gibi) servis kütüphaneleri de tasarlayabilirler.



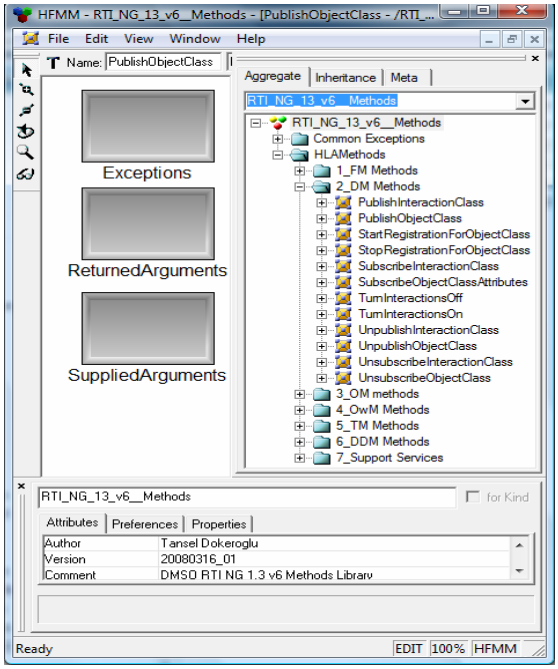
Şekil 1. ***HLA*** Servis Kütüphaneleri [8]

2.1 *DMSO HLA 1.3* Metod Kütüphanesi

DMSO HLA 1.3 Metod Kütüphanesi (*DMLib*), *DMSO HLA 1.3* uyumlu federasyonlar için federe davranışını modellemek amacıyla *DMSO RTI API*[1] metodlarını sağlar. Ekran görüntüsü Şekil 2'de sunulmuştur. Bu kütüphanede 815 adet kavram[2] tanımlanmıştır. Kavram sözcüğü ile; *DMLib*'i oluşturan dizin, atom ve model gibi *GME* stereotipleri ifade edilmek istenmiştir.

Bu kütüphaneyi Federasyon modellemesinde kullanabilmek için *DMLib*'i ilgili modele eklemek ve içindeki metodları yeri geldikçe örneklemek ya da *GME* referansları ile işaret etmek yeterlidir.

[1] *DMSO RTI NG 1.3 v6 API* için.
[2] *DMLib* versiyon 20071217_01 için.



Şekil 2. *DMSO 1.3* Metod Kütüphanesi (*GME* Ekran Görüntüsü)

3 *GME* Kütüphanesi Oluşturma Çalışmaları

3.1 *FAMM* Desteği

FAMM'ın servis desteğini sağlayan parçası, *HLA* Servis Metamodeli (*HSMM*) olarak adlandırılan kısmıdır. Tüm metod kütüphaneleri *FAMM* içerisindeki *HSMM*'e dayanmaktadır. *HSMM* oluşturulurken *IEEE* Federe Arayüz Spesifikasyonu esas alınmıştır. *HSMM*'in çeşitli standartları destekleyebileceğini sınamak amacıyla *DMSO HLA 1.3* Metod kütüphanesi geliştirme çalışması yapılmıştır. *DMSO* kütüphanesi geliştirilmesine paralel olarak *HSMM* elemanlarına eklentiler yapılmıştır. Yapılan eklentilere ileriki bölümlerde daha ayrıntılı olarak değinilecektir.

3.2 Kütüphane Geliştirilmesi

Kütüphane geliştirilirken, *HLA* Metodları ve Ortak İstisnalar *(Common Exceptions)* olmak üzere iki ana başlıkta toplanan kavramlar üzerinde çalışma yapılmıştır. Bu iki ana başlıktan öncelikle, metodlar kısmı geliştirilmiştir. *GME* ile *FAMM* kayıt *(register)* edildikten sonra *DMSO RTI 1.3 NG* standardı [9] temel kaynak olarak kullanılmış ve söz konusu dokümanın Ek-A ve Ek-B'sinde bulunan metod kütüphanesi bilgileri tek tek analiz edilmiştir. Çok sayıda metod ve argüman olduğu için, ilk aşamada bunların hepsi tablolar oluşturularak sıralanmıştır. Aralarındaki ortak noktalar nesne yönelimli bir

yaklaşımla incelenerek tespit edilmiş ve tekrar eden tipler ortaya çıkarılmıştır. Metod ve istisnalardan oluşan bu veriler daha sonraki uygulamalarda da kullanılabilecek şekilde bir veri tabanında saklanmıştır.

Bu metodların girdi *(supplied arguments)* ve çıktı *(returned types)* argümanları veri tabanına girilmiştir. Bu veri tabanında aynı isimli olup argüman ve çıktı farklılığı olan metodlar, farklılık gösterdiği kadar tekrar edilerek kütüphaneye dahil edilmişlerdir. Örneğin Federasyon Yönetimi *(Federation Management)* başlığı altında bulunan *RequestFederationSave* metodu, girdi olarak karakter dizisi ve zaman referansı olmak üzere iki argüman alarak çalışan bir metod olmanın yanında, sadece karakter dizisi alarak çalışan bir metod olma özelliğini de göstermektedir. İstisnalar *(Exception)* ile ilgili yapılan çalışmada, tüm metodlarda ortak olarak bulunan istisnaların neler olduğu tespit edildikten sonra, her metodun altına istisna sayısı kadar *exceptionreference* tipindeki argümanların referans olarak girilmesi sağlanmıştır. İstisna tanımlamalarında ek bir argüman tanımlamasına ihtiyaç duyulmamıştır.

3.3 *FAMM*'a Yapılan Eklentiler

DMLib geliştirilmesinde, analiz sonucu ihtiyaç duyulan elemanların büyük bir çoğunluğu *HSMM*'deki modelleme elemanlarından karşılanabilmiştir. Ancak özellikle argüman tanımlamalarında *DMSO HLA 1.3'e* özel birkaç ek argüman daha tanımlama ihtiyacı doğmuştur. Adı *DMSO* ile başlayan bu argümanlar kütüphaneye bu amaçla konulmuş olup Tablo 1'de sunulmuştur.

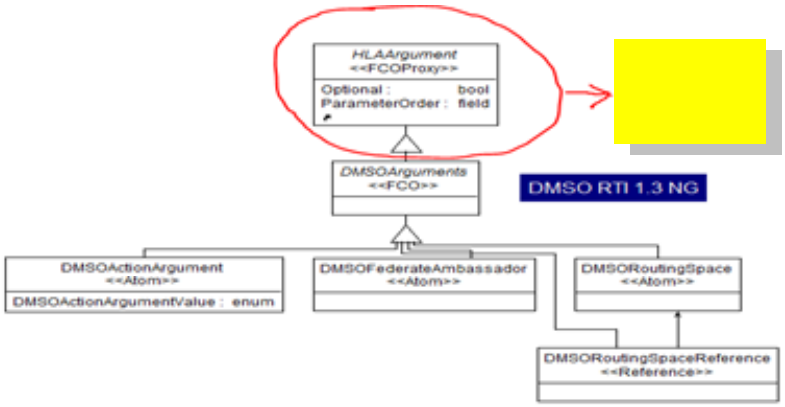
DMSOActionArgument, IEEE HLA 1516 standardındakinden [2] tamamıyla farklı olması nedeniyle yaratılmıştır. *DMSOFederateAmbassador* ve *DMSORoutingSpace* (*DMSO* Yönlendirme Uzayı) kavramları *IEEE HLA 1516* standardında bulunmamaktadır. *DMSORoutingSpaceReference* yönlendirme uzayına bir referans olarak yaratılmıştır.

Tablo 1. *DMSO RTI NG 1.3v6* için Eklenti Argümanlar [8]

ARGÜMAN	AÇIKLAMA	KULLANIM YERİ
DMSOActionArgument	Federasyondan çıkarken uygulanacak etkileri belirtmektedir.	*DMSO ResignFederationExecution*
DMSOFederateAmbassador	Federe Temsilcisinin *(Federate Ambassador)* temsilinde kullanılmaktadır.	*DMSO JoinFederationExecution*
DMSORoutingSpace	Bir Yöneltme Uzayı'nı *(Routing Space)* temsil etmektedir.	*DMSO CreateRegion*
DMSORoutingSpaceReference	Yöneltme Uzayı'na bir referanstır.	*DMSO GetDimensionName*

Bu çalışmada elde edilen sonuçlardan biri de yapılacak değişikliklerin genelde argüman seviyesinde olduğunun görülmesidir. Bu maksatla yeni argümanların tanımlanabilmesi ve mevcut argüman modeline kolayca entegre edilebilmesi için *HSMM'de HLAArgument* adlı entegrasyon noktası oluşturulmuş ve yeni *DMSO* elemanları bundan özelleştirilmiştir. İleride olabilecek "*HLA Evolved*" argümanları da buraya aynı şekilde entegre edilebilecektir..

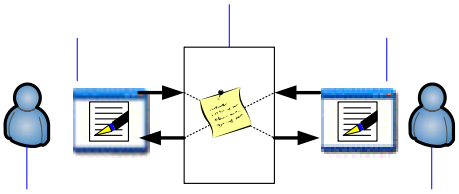
Yeni eklenen *DMSO* argümanlarının metamodel yapısı ve entegrasyon noktası Şekil 3'te gösterilmiştir.



Şekil 3. ***D****MSO* Argümanları İçin Metamodel Eklentisi

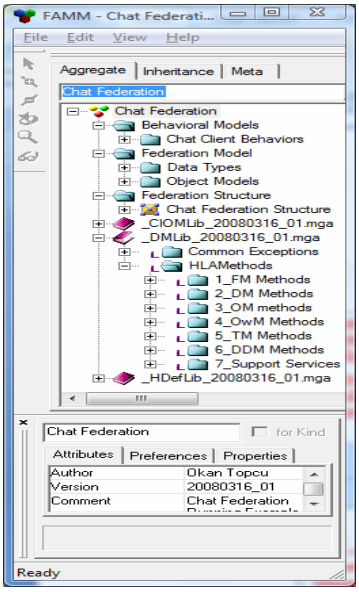
4 Kullanıcı Perspektifinden Örnek Bir Uygulama

Bu bölümde E-sohbet *(chat)* federasyonuna ilişkin federasyon mimarisinin modellemesi ve bu sırada metod kütüphanelerinin kullanımı gösterilecektir. Bu uygulamada, *HLA* altyapısının kavramlarını görmek mümkün olmaktadır. E-sohbet, *HLA* tabanlı ve dağıtık ortamda çalışan bir uygulamadır. Kullanıcılar, bir federe uygulaması olan E-sohbet uygulamasını *(chat client)* kullanarak sohbet odalarında arkadaşlarıyla metin tabanlı mesaj alıp vererek sohbet edebilmektedirler. Sohbet odalarına girmeden önce biricik bir isim seçmeleri gerekmektedir. E-sohbet uygulaması kullanıcı etkileşimi için metinsel bir arayüz sunmakta ve *RTI* iletişimini sağlamaktadır. Bu uygulama aslında, *pRTI* dağıtımının örnek bir uygulamasıdır [10]. Federasyonun kavramsal seviyedeki görünümü Şekil 4'te sunulmuştur.



Şekil 4. E-sohbet Federasyonu

Bildirinin odak noktasını gözden kaçırmamak için E-sohbet federasyon mimarisinin nasıl oluşturulacağı anlatılmayacaktır. Burada yalnızca arayüz kütüphanelerinin kullanımı vurgulanacaktır. E-sohbet Federasyon Mimarisinin bir *GME* projesi olarak görünümü Şekil 5'te sunulmuştur.

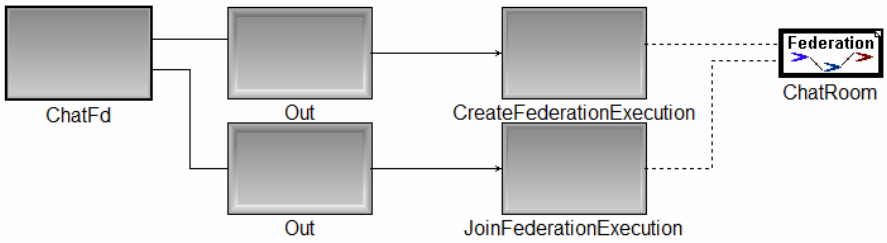


Şekil 5. E-sohbet Federasyon Mimarisi (GME Ekran Görüntüsü)

Şekil 5'te görüldüğü gibi ana proje klasörü *(Chat Federation)* altında federasyon mimarisini oluşturan Davranış Modelleri *(Behavioral Models), FOM* ve *SOM*'ların tanımlandığı Federasyon Modelleri *(Federation Models)* ve federasyon statik yapısının tanımlandığı Federasyon Yapısı *(Federation Structure)* klasörleri bulunmaktadır.

Federasyon mimarisi, federasyon tasarımcısının bakış açısı göz önüne alınarak detaylı olarak [4]'de anlatılmıştır.

Federe davranış modelleri, *RTI* ile olan etkileşimin modellenebilmesi amacıyla *HLA* metodlarının *LSC*'ler içinde bir mesaj gibi çağrılmasına olanak sağlamaktadır. Bunun için kullanılabilecek olan *DMSO RTI* metodları ve istisnaları *DMLib*'te bir şablon görevi görecek şekilde tanımlanmıştır. Şekil 5'te görüldüğü gibi *DMLib,* projeye bir *GME* kütüphanesi (kitap şeklindeki ikon) olarak eklenmiştir. *DMLib'*te tanımlanmış olan bu metodlar Şekil 6'da görüldüğü gibi bir federenin federasyona katılış davranışını temsil eden federe davranış modelleri içinde kullanılabilmektedir. Buradaki *CreateFederationExecution* ve *JoinFederationExecution* metodları *DMLib*'teki şablonlardan yaratılarak kullanılmıştır.



Şekil 6. E-sohbet Federesinin Federasyona Katılış Davranış Modeli

Şekil 6'da *ChatFd*, bir federeyi, *ChatRoom* ise bir federasyon koşumunu *(federation execution)* temsil etmektedir. *Out* olayları *(event)* ise *ChatFd'*den *ChatRoom*'a mesaj gönderilmesini temsil etmektedir.

Şekil 5'te görülen Konsol Giriş Çıkış Kütüphanesi *(CIOMLib)* ve *HLA* Standard Değerleri Kütüphanesi *(HdefLib)* ile ilgili detaylar [4]'den elde edilebilir.

5 Sonuç

Bu çalışma ile *DMSO HLA 1.3* uyumlu federasyonlar için yeni bir federe arayüz spesifikasyon kütüphanesi oluşturulmuştur. Yapılan çalışma ile *FAMM*'a *DMSO HLA 1.3* desteği kazandırılmış ve olabilecek bu tip genişletmelerin kolaylaştırılması için metamodelde entegrasyon noktaları yaratılmıştır.

GME'nin kütüphane mekanizmasının kullanılması da metamodelin esneklik kazanmasını sağlamıştır. Adı geçen tüm metamodellere, kütüphanelere ve örnek uygulamalara *FAMM'*ın web sitesinden [11] ulaşılabilir.

Bu çalışmanın devamı kapsamında, model (federasyon mimarileri) bazında *IEEE HLA 1516* ve *DMSO HLA 1.3* federasyon mimarilerinin birbirlerine otomatik dönüştürülmesi mümkün görülmektedir. Eski federelerin, *IEEE HLA 1516* standardına aktarılabilmesi için böyle bir olanağın yararlı olacağı düşünülmektedir.

Kaynakça

1. IEEE 1516 Standart for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules, September 2000.
2. IEE 1516.1 Standart for Modeling and Simulation (M&S) High Level Architecture (HLA)- Federate Interface Specification, September 21, 2000.
3. IEEE 1516.2 Standart for Modeling and Simulation (M&S) High Level Architecture (HLA) Object Model Template Specification, 21 September 2000.
4. Okan Topçu, "Metamodeling for the HLA Federation Architectures", Doktora Tezi. Ortadoğu Teknik Üniversitesi, Bilgisayar Mühendisliği, 25 Aralık 2007.
5. Mehmet Adak, "Model-based Code Generation for HLA Federates", Doktora Tezi. Ortadoğu Teknik Üniversitesi, Bilgisayar Mühendisliği, 2007.
6. Gürkan Özhan ve Halit Oğuztüzün, "Model-integrated Development of HLA-based Field Artillery Simulation", European Simulation Interoperability Workshop (Euro-SIW), 19-22 Haziran 2006.
7. Okan Topçu, Mehmet Adak, ve Halit Oğuztüzün, "A Metamodel for Federation Architectures", ACM Transactions on Modeling and Computer Simulation (yayınlanacak, Temmuz 2008).
8. ISIS, "A Generic Modeling Environment GME 6 User's Manual v6.0". Institute for Software Integrated Systems (ISIS) Vanderbilt University, 2006.
9. DMSO, "RTI1.3 – Next Generation Programmer's Guide", U.S. Department of Defense, Defense Modeling and Simulation Office, 2002.
10. Pitch RTI Web Sitesi, "http://www.pitch.se/products/pitch-prti/pitch-prti-overview.html",en son 13 Nisan 2008 tarihinde erişilmiştir.
11. FAMM Web Sitesi, "http://www.ceng.metu.edu.tr/~otopcu/famm/", en son 13 Nisan 2008 tarihinde erişilmiştir.

Komuta Kontrol Sistemlerinde Alan Modeli ve Referans Mimari Kullanımı

Tankut Koray[1], Cihangir Tolga Yurdakul[2], İskender Yakın[3]

[1,2,3] Aselsan A.Ş. SST-MD-YMM, 06172, Yenimahalle, Ankara
[1] tkoray@aselsan.com.tr, [2] ctolga@aselsan.com.tr, [3] iyakin@aselsan.com.tr

Özet. Alan modeli, geliştirme yapılan uzmanlık alanı ile ilgili kavram ve yeteneklerden oluşmaktadır. Referans mimari ise alan modelini uygulanabilir bileşenler olarak ifade etmektedir. Komuta kontrol sistemlerinde, alan modeli ve bu modele dayalı referans mimari kullanılarak uygulama mimarisi türetmek geliştirme sürecinde avantaj sağlamaktadır. Türetilen uygulama mimarisine uygun hazır altyapılar kullanılarak çevik ve etkin bir şekilde yazılım geliştirilmesi mümkün olmaktadır. JAUS (Joint Architecture for Unmanned Systems), insansız sistemler alanında ortak bir alan modeli ve buna bağlı bir referans mimari tanımlamaktadır. Bu bildiride ASELSAN'da geliştirilen insansız sistemler için uygulama mimarisi oluşturma sürecinde JAUS ve uyumlu teknik altyapılar ile elde edilen deneyim ve kazanımlar aktarılmaktadır.

1 Giriş

ASELSAN Savunma Sistem Teknolojileri (SST) Grubu, Hava Savunma, Ateş Destek, Anayurt Güvenliği ve İnsansız Sistemler alanlarında çok sayıda komuta kontrol sistemi geliştirmektedir. Bu alanların toplamı K4İGK (Komuta, Kontrol, Komünikasyon, Kompüter, İstihbarat, Keşif ve Gözetleme) alanı olarak ifade edilir. K4İGK alanında gerek geliştirilen yazılım sayısının fazlalığı ve karmaşıklığı, gerekse kısa zamanda ihtiyaca cevap verme zorunluluğu özellikle yazılım unsurları arasında yeniden kullanımı gerektirmektedir.

K4İGK alanı birden fazla ve birbiriyle doğrudan ilişkili olmayan alt uzmanlık alanlarından oluşur. Bir uzmanlık alanında yazılım geliştirebilmenin ön koşulu ilgili alan hakkında bilgi sahibi olmaktır. Alan mühendisliği bu ihtiyaca cevap verir. Alan mühendisliğinin amacı alan bilgisinin ihtiyaca yönelik olarak ifade edilmesi ve yeniden kullanılabilir bileşenlerin tanımlanmasıdır. Alan bilgisinin yazılım geliştirme aşamasında yeniden kullanılabilir bir şekilde ifade edilmesine alan modeli, alan modeli kullanılarak tanımlanan yeniden kullanılabilir bileşenlerin oluşturduğu mimariye teknik referans mimari denir.

Yazılım mühendisliğinde birçok ürün yeniden kullanılabilir. Bu ürünler yazılım gereksinimleri, mimariler, tasarım, tasarım kalıpları, kod veya test altyapısı olabilir. Alan modeli uzmanlık alanındaki kavramları, ihtiyaçları ve alanın kapsamını ortaya koyar. Komuta kontrol sistemlerinde genellikle çeşitli yetenek seviyelerinde birçok yazılım bir arada çalışır. Bu yazılımlar, temelde aynı uzmanlık bilgisi ışığında geliştirilir. Bu açıdan bakıldığında alan modelinin yeniden kullanıldığı söylenebilir. Aynı amaca yönelik teknik referans mimarinin çeşitli seviyelerdeki yazılımlarda

yeniden kullanılması daha üst seviyeli bir kullanımdır. Bu aşamada bileşen seviyesinde yeniden kullanım söz konusudur. Teknik referans mimarinin yeniden kullanılmasıyla yazılım geliştirme hızı, yazılım kalitesi ve yazılımlar arasındaki benzerlik önemli derecede artmaktadır. Bu şekilde geliştirilen yazılımların toplamına yazılım ailesi denir.

Alan modelinin diğer bir amacı geliştirici ile kullanıcı arasında ortak bir dil oluşturmaktır. Alan modelini oluşturan alan uzmanlarına kullanıcılar da dahil olabilmektedir. Alan modeli yeni bilgiler, kapsam ve eğilimler doğrultusunda güncellenmektedir. Bu sayede geliştirici ile kullanıcı arasındaki dil yaşamakta ve gelişmektedir.

Askeri alandaki alan modeli bilgi kaynakları arasında talimnameler, yönergeler, alan uzmanı kullanıcı ve geliştiriciler, mevcut sistemler, yazılımlar ve benzer amaçlara yönelik hazırlanmış diğer alan çalışmaları yer alır. Literatürde tanımlı alan modelleri ve referans mimarilerinden bazıları POSIX [1], JTA (Joint Technical Architecture) [2], WSTAWG-OE (Weapon Systems Technical Architecture Working Group Operating Environment) [3], Raytheon COE [4], FC-NET (Fire Control Node Engagement Technology) [5], Boeing OEP (Boeing Open Experimental Platform) [6], ARGOS [7], Joint Architecture for Unmanned Systems (JAUS) [8] çalışmalarıdır.

Bahsedilen alan modelleri ve referans mimariler internet üzerinden paylaşılmakta, bu sayede dağıtık bir geliştirme ortamı da sağlanmaktadır. Çalışmalara, ilgili alanlarda geliştirme yapan büyük şirketler ve organizasyonlar katılmakta, çıktılar bu sayede birçok insan tarafından gözden geçirilmekte, beslenmekte ve olgunlaşmaktadır. Bu sayede herkes tarafından aynı şekilde anlaşılmış bir uzmanlık altyapısı oluşturulmakta, farklı kişiler tarafından geliştirilen yazılımlar arasındaki karşılıklı uyumlu çalışabilirlik desteklenmektedir.

Bildiride insansız sistemler alanı için tanımlanmış JAUS alan modeli ve referans mimarisi kullanılarak yazılım geliştirme tecrübeleri aktarılacaktır. JAUS kullanım tecrübesinin aktarılması ile yazılım geliştirilen alandaki bilginin yeniden kullanımının önemi vurgulanacaktır.

2 JAUS Alan Modeli ve Referans Mimarisi

JAUS, insansız sistemler için tasarlanan ortak bir alan modeli ve referans mimaridir. İnsansız sistemler alanında bir açık mimari (open architecture) geliştirmek amacıyla 1995 yılında JAUGS WG (Joint Architecture for Unmanned Ground Systems Working Group) tarafından başlatılmış ve A.B.D. Savunma Bakanlığı tarafından onaylanmış bir programdır. İlk olarak sadece insansız kara araçları için başlatılan bu program daha sonra tüm insansız araçlar için ortak bir mimari geliştirme amacına uygun olarak genişletilmiştir.

İnsansız sistem geliştirme sürecini hızlandırmak ve geliştirilen sistemin JAUS uyumlu olduğunu temin etmek amacıyla JAUS alan modeli ve referans mimarisi tanımlanmıştır. JAUS alan modeli, JAUS gereksinimlerini tanımlamak amacıyla oluşturulmuştur ve insansız sistem fonksiyonları ve bilgilerini tanımlar [9]. JAUS referans mimarisi ise JAUS uyumlu bir insansız sistem geliştirmek için gerekli olan teknik ayrıntıları içermektedir. Mimaride sunulan bu teknik ayrıntılar sistem kabulü sürecinde yeterlilik

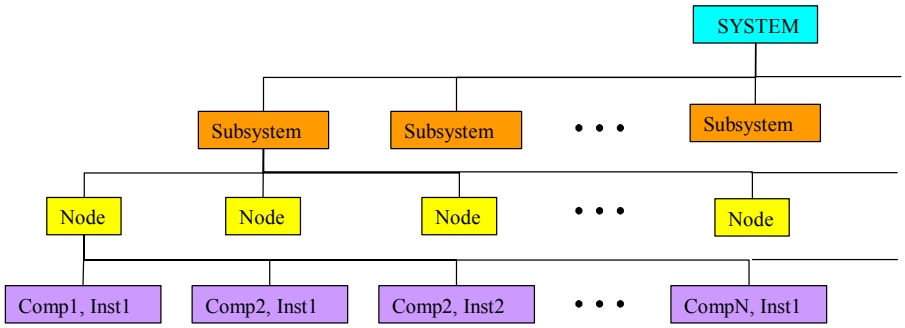
ve JAUS uyumluluk ölçütleri olarak kullanılmaktadır [10]. JAUS referans mimarisinin temel amacı yeni insansız sistem bileşenleri tasarlamada kullanılan tüm fonksiyon ve mesajların nasıl tanımlanması gerektiğini göstermektir. İnsansız sistem geliştirme sürecinde JAUS alan modeli ve mimarisi kullanılması durumunda elde edilmesi beklenilen kazanımlar [11],

- Sistem hayat döngüsü masraflarının azalması,
- Geliştirme ve entegrasyon sürelerinin kısalması,
- Yeni teknolojinin var olan sistemlere hızlı bir şekilde eklenebilmesi,
- Var olan sistemlere yeni yetenek eklenmesinin kolaylaştırılmasıdır.

JAUS belirli görevleri yerine getirmek için çalışan düğümler (node) arasındaki haberleşme yöntemleri ve mesaj formatlarını tanımlayan, bileşen tabanlı bir mimaridir [8]. Günümüz ve geleceğin tüm insansız sistemlerine uygulanabilir olması için, beş prensip doğrultusunda geliştirilmektedir [9]. Bunlar,

- Araç platformundan bağımsız olma,
- Görevlerden bağımsız olma,
- Bilgisayar donanımından bağımsız olma,
- Teknolojiden bağımsız olma
- Operatör kullanımından bağımsız olmadır.

Şekil 1'de görüldüğü gibi mimari, alt sistemler, düğümler ve bileşenler üzerine kurulu hiyerarşik sistemlerin tanımlanmasına yardımcı olur. Yüksek seviyede birlikte çalışabilirliği sağlamak amacıyla olası alt sistem, düğüm ve bileşenler arasındaki mesaj kümelerini tanımlar. Alt sistem, düğüm ve bileşen tanımları dahil, mimarinin önemli bir bölümü yukarıda bahsedilen beş prensibe bağlı kalınarak esnek bir şekilde tanımlanmıştır.



Şekil 1. JAUS sistem yapısı

Bu prensipler doğrultusunda JAUS, tüm insansız sistem çeşitlerini desteklemeyi, hızlı teknoloji kullanımını, birlikte çalışabilir komuta kontrol konsolları geliştirmeyi, birlikte çalışabilir ve değiştirilebilir faydalı yük kullanımını ve birlikte çalışabilir insansız sitemler geliştirmeyi hedeflemektedir.

JAUS mimarisi hükümet, endüstri ve üniversitelerden oluşan JAUS Çalışma Grubu tarafından geliştirilmektedir. Mimari şu anda Otomotiv Mühendisliği Birliği Açık Mimari Çerçevesi kapsamında SAE AS-4 adı altında standart haline getirilme sürecindedir.

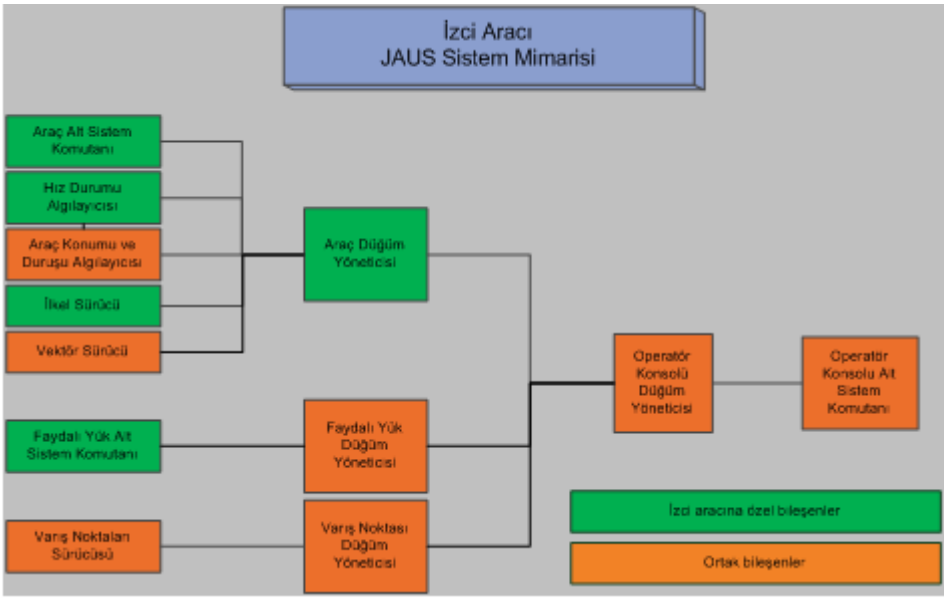
3 Komuta Kontrol Sistemlerinde Kullanım Tecrübeleri

ASELSAN'da insansız sistemler kapsamında prototip olarak hazırlanmış iki tane kara aracı geliştirilmiştir; İzci ve Gezgin. İlk olarak geliştirilen İzci, üzerine çeşitli faydalı yükler takılabilen, orta ölçekli ve tekerlekli bir kara aracıdır. Gezgin ise üzerine çeşitli faydalı yükler takılabilen küçük-orta ölçekli ve paletli bir kara aracıdır. Ayrıca Denizci isimli insansız su üstü aracının prototip geliştirme çalışmaları devam etmektedir. Bu üç sistem Şekil 2'de görülmektedir. Bu sistemlerin ve ilerde geliştirilmesi düşünülen diğer insansız sistemlerin komutası için ortak bir operatör konsolu geliştirilmiştir. Operatör konsolu ile aynı anda birden fazla aracın kontrolü yapılabilmektedir. Geliştirilen sistemler ile operatör konsol yazılımı arasındaki birlikte çalışabilirlik JAUS ile sağlanmaktadır.

Şekil 2. ASELSAN tarafından geliştirilen İzci, Gezgin, Denizci araçları

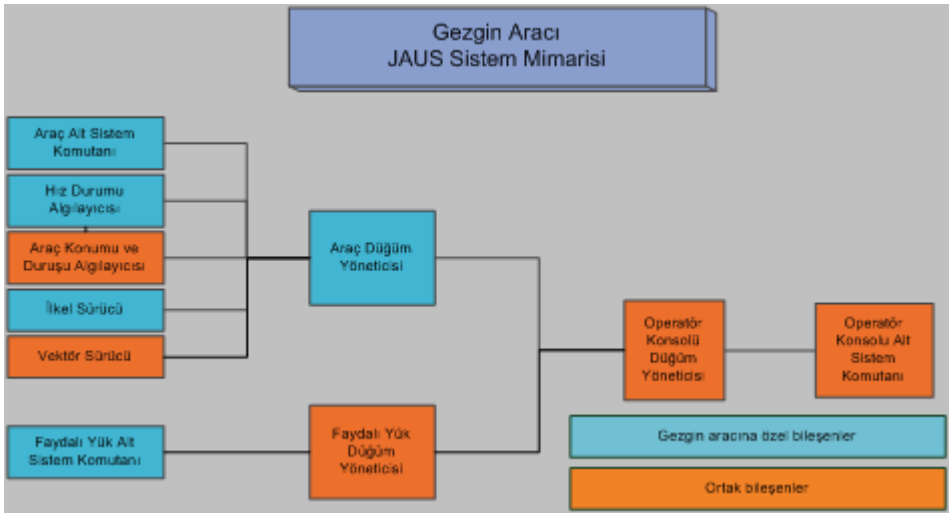
JAUS, araç kontrolü için "İlkel Sürücü" ve "Vektör Sürücü" bileşenlerini tanımlamaktadır."İlkel Sürücü" basit sürüş ve mobilite fonksiyonları içeren JAUS bileşenidir. "Vektör Sürücü" ise belirtilen yön, hız ve yükseklik isteklerine göre sürüş yapan bileşendir. İzci aracının kontrolü için "İlkel Sürücü" ve "Vektör Sürücü" düğümleri gerçeklenmiştir. Faydalı yüklerin kontrolü için "Faydalı Yük Alt Sistem Yöneticisi" gerçeklenmiştir. İzci aracının sistem mimarisi Şekil 3'de görülmektedir.

Mimaride yer alan düğümler, ilgili donanım kontrol yazılımları ile birlikte çalışmaktadır.



Şekil 3. İzci aracı JAUS mimarisi

Daha sonra geliştirilen Gezgin aracının sistem mimarisi, İzci aracının mimari altyapısı kullanılarak oluşturulmuştur. Gezgin aracı için, İzci aracına özel olarak geliştirilen bileşenler yerine JAUS uyumlu yeni bileşenler geliştirilmiştir. Yeni bileşenler Gezgin aracının donanım arayüzlerini uyumlandırmıştır. JAUS uyumluluğunun değişmemesi sebebiyle İzci 'nin araç kontrol yazılımı ile operatör konsolu aynen korunarak Gezgin aracının kontrolü sağlanmıştır. Bu şekildeki bir yeniden kullanım sayesinde yeni araçların kontrolü ve uyumlanması için ihtiyaç duyulan süre çok kısaltılmıştır. Gezgin aracının sistem mimarisi Şekil 4'de görülmektedir.



Şekil 4. Gezgin aracı JAUS mimarisi

JAUS alan modeli, insansız sistemler geliştirme çalışmalarının ilk aşamalarında ekip elemanlarının insansız sistemler hakkında belirli bir bilgi seviyesine gelmesi, alan hakkında genel bilgi sahibi olması ve insansız sistemler alanının kapsamının ve çerçevesinin anlaşılması açısından çok faydalı olmuştur. Ekip elemanlarının alan hakkında uzmanlaşma sürecini hızlandırmıştır. Kavramların herkes tarafından aynı şekilde anlaşılmasını sağlamıştır. Özellikle yazılım ve sistem mühendisliğinin birlikte yaptığı çalışmalarda bu ortak anlayış geliştirme sürecini hızlandırmıştır.

Alan modelinin uygulamaya geçirilmesinde, JAUS referans mimarisi hızlı bir başlangıç yapmaya imkân tanımıştır. Referans mimari kapsamında ortaya konan bileşenler ve mesajlar daha fazla tanıma gerek duymadan gerçeklenebilmiştir. Böylece hızlı bir JAUS alt yapısı kurulabilmiş ve denemeler yapılabilmiştir. Daha sonra kullanılan çerçeveler ile desteklenen bu yapı, yazılımların temel mimarisini oluşturmuştur. JAUS mesaj setini ve bu set ile IP tabanlı haberleşme sağlayan açık kaynak kodlu OpenJAUS [12] çerçevesi kullanılmıştır. Bu çerçeve içerisinde JAUS mesaj setini hem C/C++ hem de Java gerçeklemeleri bulunmaktadır.

JAUS referans mimarisi, servis yönelimli bir mimaridir; JAUS bileşenleri ve bu bileşenlerin sağladıkları servisler olarak tanımlanmıştır. Herhangi bir JAUS bileşeni başka bir bileşenin kontrolünü alıp servislerini kullanabilmektedir. Böylelikle JAUS, sistem yapılandırmaları üzerine hiçbir kısıtlama getirmemektedir. Böyle bir mimariyi, gerçekleştirmek için açık kaynak kodlu OSGi (Open Services Gateway initiative) Servis Platformu [13] kullanılmıştır. Java'ya dinamik ve modüler bir bileşen modeli getiren OSGi Servis Platformu ile servis yönelimli mimariler kolaylıkla uygulanabilmektedir. JAUS bileşenleri, servis tabanlı OSGi bileşenleri olarak paketlenmiştir. Böylelikle, sistem yapılandırılmasına uygun bileşenler, OSGi Servis Platformu'na dahil edilerek, sağladıkları servisler kullanılabilmektedir.

Geliştirilen komuta kontrol konsolunun birden fazla ve farklı tipte aracı kontrol etmesi amaçlanmıştır. Kullanıcıya, kontrol edilen her bir araç için farklı kamera görüntüleri, araç ve faydalı yük bilgileri, görev durumları gösterilmelidir. Böylesine zengin bilgi sağlayan bir yazılımın etkin olarak kullanılabilmesi için kullanıcı dostu ve uyarlanabilir olması önemlidir. Farklı araçlara özel veya uzak komuta, otonom sürüş gibi kullanım durumlarına yönelik farklı görünümlerin sağlanması, kullanıcı tercihlerinin saklanabilmesi, farklı yetenek yapılanmalarının desteklenmesi gerekmektedir. Bahsedilen yeteneklerin sağlanması için açık kaynak kodlu "Eclipse Zengin İstemci Platformu" kullanılmıştır. Her JAUS bileşeni için bir Eclipse RCP kullanıcı arayüzü elemanı tasarlanmıştır. Benzer yetenekleri sağlayan kullanıcı arayüzü ve JAUS bileşenlerinin birlikte paketlenmesi ile kontrol edilen sistemin kabiliyetlerine ve gerekli olan JAUS bileşenlerine göre uygun görünümlerin açılıp kapatılabilmesi mümkün olmuştur. Ayrıca kontrol edilen araç değişse bile JAUS tarafından tanımlanan yazılım arayüzleri aynı kaldığı için yazılımda değişiklik yapılmadan JAUS uyumlu farklı insansız sistemler de kontrol edilebilmiştir. Ayrıca, "Eclipse Zengin İstemci Platformu" sayesinde kontrol edilen sisteme özel görünümler de konsola kolaylıkla eklenebilmiştir.

Yazılım geliştirilen alanın teknik ihtiyaçlarına cevap veren hazır çerçevelerin kullanılması sayesinde hızlı prototipleme yapılabilmiştir. Ayrıca çerçevelerin genişlemeye ve değişime uygun yapıları sayesinde, gelen yeni istekler ya da değişiklikler kısa sürede uygulanmıştır.

Alan modelinde olup referans mimarisinde daha yerini alamamış kavramların da gerçeklenmesine ihtiyaç duyulmuştur. Özellikle faydalı yük ile ilgili bileşenlerin daha tanımlanmamış olması nedeni ile silah sistemleri, meteorolojik algılayıcı vb. faydalı yükler için bileşenler tanımlanmıştır. Bu bileşenler ile ilgili JAUS uyumlu mesajlar tanımlanmış ve JAUS yapısına dâhil edilmiştir. Başka bir deyişle diğer tanımlı JAUS bileşenleri ve mesajları gibi özelleşmiş bileşenler ve mesajlar tanımlayıp mimari genişletilebilmektedir.

4 Sonuç

Bildiride insansız sistemler alanında alan bilgisi ve referans mimarisinin yeniden kullanılması ile elde edilen tecrübe ve kazanımlar aktarılmıştır. ASELSAN SST Grubu Yazılım Mühendisliği Müdürlüğü yazılım yeniden kullanılabilirliğinin önemini kavramış, geçmiş deneyimleri kullanarak ve akademik işbirliği içerisinde K4İGK ve Silah Sistemleri için Alan Analizi Yapılması ve Referans Yazılım Mimarisi Geliştirilmesi amacıyla iki ayrı çalışma yürütmektedir. Bu çalışmalar sonucunda günümüze kadar uygulanan yeniden kullanımın daha etkin ve kalıcı bir şekilde devam ettirilmesi hedeflenmektedir.

Kaynakça

1. IEEE Std. 1003.1, 2004 Edition, The Open Group Technical Standard Base Specifications, Issue 6, 30 April 2004.
2. Department of Defense Joint Technical Architecture Version 6.0, 2 October 2003.
3. Adams, C., "Weapon System Technical Architecture Working Group" , Presentation, 1996.

4. Hudson, J.C., “Rayheon Common Operating Environment”, 2006.
5. Sherrill, J.,O’Guin, R., Butler, D.A, “Fire Control-Node Engagement Technology (FC-NET) A Distributed Technical Fire Control Software Architecture”.
6. Sharp, D., “Weapon System OEP Breakout Session”.
7. Nijenhuis, M., “Characteristics of a Modern Combat Management System Architecture”.
8. JAUS, Reference Architecture Specification, Volume 2, Part 2, Message Definition. Version 3.3, June 27, 2007.
9. JAUS, Domain Model, Volume 1. Version 3.2., 10 March 2005.
10. JAUS, Reference Architecture Specification, Volume 2, Part 1, Architecture Framework. Version 3.3, June 27, 2007.
11. Jorgen Pedersen, President & CEO, RE2, Inc. A Practical View and Future Look at JAUS. May 2006.
12. OpenJAUS, http://www.openjaus.com.
13. Open Services Gateway Initiative, http://www.osgi.org.

Sınıfların Karmaşıklık Ölçütleri Kullanılarak Tekrar Tasarım Durumlarının İrdelenmesi

Burak Turhan[1], Ayşe Bener[2], Yasemin Köşker[3]

[1, 2, 3] Boğaziçi Üniversitesi, Bilgisayar Mühendisliği Bölümü, 34342, Bebek, İstanbul
{[1] turhanb, [2] bener, [3] yasemin.kosker}@boun.edu.tr

Özet. Yazılım hayat döngüsünde geliştirme maliyetleri buzdağının sadece görünen kısmını oluşturur. Maliyetin büyük kısmı hata düzeltme ve iyileştirme amaçlı bakım çalışmalarına harcanır. Yazılımlar değişikliğe uğradıkça yapısallıkları büyük ölçüde bozulur ve karmaşıklıkları artmaya başlar [1]. Tekrar tasarım (refactoring) bu amaçla yazılım kalitesinin korunmasını sağlayan ve bakımı kolaylaştıran bir yaklaşımdır. Bu bildiride sınıf karmaşıklık ölçütlerini kullanarak yapay öğrenme teknikleri ile yeniden tasarım gerektiren sınıfların belirlenmesi amaçlanmaktadır. Önceki çalışmalarımızda hata tahmini için kullandığımız ve yüksek performans elde ettiğimiz Bayes sınıflandırıcıları bu çalışmamızda tekrar tasarım tahmini için önerilmiştir. Deneylerimizde, önde gelen bir GSM şirketinin alt yapısını oluşturan yazılımlardan topladığımız 4 versiyona ait karmaşıklık ölçütleri kullanılmıştır. Sonuçlarımız tekrar tasarım gerektiren sınıfların ortalama %82'sini doğru tahmin edebildiğimizi ve bunu yaparken sadece %13 oranında yanlış yönlendirme yapıldığını göstermektedir. Sonuçlarımız tekrar tasarım çalışmaları sırasında yazılım mimarlarına ve geliştiricilere vakit kazandıracak ve bu değerli insan kaynağından daha verimli faydalanılmasını sağlayacaktır.

1 Giriş

Projelerin büyüklüğü artıp yazılım üzerinde fonksiyonel iyileştirmeler yapıldıkça, yazılımın karmaşıklığı da artar ve dolayısıyla bakımı zorlaşır. Tekrar tasarım, yazılımın dışarıdan gözlenen davranışını değiştirmeden tasarımının iyileştirilmesini amaçlayan bir yaklaşımdır. Tasarımda yapılan iyileştirme sonrası yazılımın belli girdilere karşılık ürettiği çıktılar, değişiklik öncesi gözlemlenen girdi-çıktılarla aynı olmalıdır [2]. Bu nedenle istenilen işlevi doğru bir şekilde yapan yazılımların tekrar tasarımının yapılması dikkat isteyen, zaman ve deneyim gerektiren bir iştir. Tekrar tasarım başlıca 3 aşamadan oluşur [3]:

- Tekrar tasarım gerektiren kod bölümlerinin tespit edilmesi,
- Potansiyel tekrar tasarımların fayda analizinin yapılması,
- Uygun görülen tekrar tasarımların uygulanması.

Bu bildiride deneyimli ve nitelikli yazılımcıların zaman harcamasına ihtiyaç duyulan, tekrar tasarım gerektiren kod bölümlerinin tespit edilmesi aşamasına odaklanılmıştır. Bu aşamada yardımcı araçlar kullanılması ilgili yazılımcıları doğru kod parçalarına

yönlendirmekle birlikte zaman ve maliyetleri de azaltacaktır. Tekrar tasarım uygulanması, karmaşıklığı yüksek olan sınıf ya da metodların daha basit ve anlaşılır şekilde yeniden kodlanmasını sağlayıp, bu sayede daha az deneyime sahip geliştiricilerin bu kodların bakımında görev alabilmesini de mümkün kılacaktır.

Bu bildiride yukarıda saydığımız amaçlara yönelik olarak tekrar tasarım gerektiren sınıfların otomatik olarak tespit edilmesine yönelik bir model önerilmiştir. Tekrar tasarım bir yapay öğrenme problemi olarak ele alınmış ve yazılım projelerinde tekrar tasarımı yapılmış sınıfların karmaşıklık karakteristikleri modellenerek [4], daha sonra üretilen veya iyileştirme yapılan sınıflardan tekrar tasarım gerektirenler tespit edilmeye çalışılmıştır. Daha önce yazılım hata tahmini çalışmalarımızda kullandığımız Ağırlıklandırılmış Bayes modelini kullanarak, tekrar tasarım problemi bir sınıflandırma (classification) problemi olarak tanımlanmıştır [5]. Önerdiğimiz model bir GSM firmasının altyapısını oluşturan yazılımlar üzerinde sınanmış ve modelin %82 oranında doğru tahminler ürettiği gözlemlenmiştir.

Bildirinin 2. bölümünde önerilen modeller ayrıntılı olarak açıklanmıştır. 3. bölümde deney tasarımı ve sonuçlar sunulmuştur. 4. bölümde ise tartışmaya yer verilmiştir.

2 Kullanılan Modeller

Bayes modelleri veri analizinde kullanılan temel tekniklerden biridir. Kullanımını önerdiğimiz model olan Ağırlıklandırılmış Bayes modeli, standart Bayes modelinin genelleştirilmiş bir halidir [5, 6, 7, 8, 9, 10, 11]. Her iki model de hata tahmini kapsamında daha önceki çalışmalarımızda kullanılmış ve yüksek yakalama oranları ile düşük yanlış uyarı oranları sergilemişlerdir (Performans ölçüteri Bölüm 3.2'de açıklanacaktır). Bu sebeple tekrar tasarım tahmini probleminde de kullanılmaları uygun görülmüştür.

Standart Bayes modeli belli varsayımlara dayanmaktadır. Bunlardan birincisi modelde kullanılan girdilerin birbirinden istatistiksel olarak bağımsız olduklarıdır [6, 7]. İkincisi ise tüm girdilerin eşit öneme sahip olduğu varsayımıdır [5, 8]. Bu bildiride ilk varsayım olduğu gibi kabul edilmiş ancak ikinci varsayımı yapmayan Ağırlıklandırılmış Bayes modeli de kullanılmıştır. Ağırlıklandırılmış Bayes modeli ikinci varsayımı yapmadığı için modelin girdilerine karşılık gelen ağırlıkların da öğrenilmesini gerektirmektedir. Bu aşamada da karar ağaçlarının öğreniminde kullanılan InformationGain algoritmasının sağladığı girdi önem sıraları ağırlıklara dönüştürülerek kullanılmıştır [10, 11].

2.1 Naive Bayes Modeli

Tanımlanan problemde amaç tekrar tasarım gerektiren ve gerektirmeyen kod sınıflarını:

- C0: Tekrar tasarım gerektirmiyor
- C1: Tekrar tasarım gerektiriyor

şeklinde sınıflandırmaktır. Bayes teoreminden elde edilen Naive Bayes sınıflandırıcısı basit ancak etkili bir modeldir. Bayes teoremi, sonsal dağılımın önsel dağılım ve olabilirlikle orantılı olduğunu söyler ve Denklem 1'deki şekilde ifade edilir [11]:

$$P(C_i \mid x) = \frac{P(x \mid C_i)P(C_i)}{P(x)} \quad (1)$$

Paydada bulunan terim bütün örnekler için ortak olduğundan göz ardı edilebilir. Önsel dağılım olarak, yapılan varsayımlar gereği, tek değişkenli normal dağılım kullanılılır. Karmaşıklığı azaltmak için, her iki sınıfa ait örneklerin tek bir ortak değişinti matrisini paylaştıkları varsayımı yapılabilir. Paylaşılan ortak değişinti matrisi, her sınıfın kendi ortak değişinti matrislerinin, sınıfların önsel olasılıkları ile ağırlıklandırılmaları ile elde edilebilir. Bu ortak değişinti matrisinin ana köşengen olmayan tüm üyelerinin sıfır (0) olduğu varsıyımı yapılarak Naive Bayes sınıflandırıcısı elde edilir. Naive Bayes sınıflandırıcısı Formül 2'deki şekilde belirtilir [11].

$$g_i(x) = -\frac{1}{2}\sum_{j=1}^{d}\left(\frac{x_j^t - m_{ij}}{s_j}\right)^2 + \log(P(C_i)) \quad (2)$$

Pratik kullanımda, tekrar tasarımı yapılmamış (C0) ve tekrar tasarımı yapılmış (C1) kod örneklerinin ölçümleri üzerinden Formül 2'de belirtilen model parametreleri (*m, s*) gözetimli şekilde öğrenilir. Sonradan geliştirilen kod örneklerinden aynı ölçümler alınarak öğrenilmiş modele girdi olarak verilir ve modelin yaptığı tahmin çerçevesinde ilgili kodun tekrar tasarım gerektirip gerektirmediği belirlenir.

2.2 Ağırlıklandırılmış Bayes Modeli

Naive Bayes modelinin varsayımlarından birisi, önceden belirtildiği gibi, modelin tüm girdilerinin eşit öneme haiz olduğu varsayımıdır. Ağırlıklandırılmış Bayes modelinde ise bu tür bir varsayım yapılmaz ve girdilere otomatik olarak ağırlıklar atanır. Bu durumda Formül 2'ye bir ağırlık terimi (*w*) eklenir ve Formül 3'teki şekilde yeniden yazılır [5, 8]:

$$g_i(x) = -\frac{1}{2}\sum_{j=1}^{d} w_j\left(\frac{x_j^t - m_{ij}}{s_j}\right)^2 + \log\left(P\left(C_i\right)\right) \quad (3)$$

Girdilerin ağırlıklarının kestirimi için bu çalışmada InformationGain algoritması kullanılmıştır. InformationGain algoritması eldeki sınıflandırma problemi için en iyi ayrım gücüne sahip girdileri sıralamak amacı ile kullanılır. En çok kullanıldığı alan karar ağaçlarının öğrenilmesinde dallanma yapılacak noktaların belirlenmesidir. Bu çalışmada InformationGain algoritmasının sıralama amaçlı ürettiği performans kriteri normalizasyon sonucunda girdi ağırlıklarına dönüştürülmüştür [5].

Pratik kullanımda, tekrar tasarımı yapılmamış (C0) ve tekrar tasarımı yapılmış (C1) kod örneklerinin ölçümleri üzerinden (elimizde tekrar tasarım verisi bulunmadığı için bu ölçümler 3.1'deki varsayıma dayanılarak üretilmiştir) Formül 3'de belirtilen model parametreleri (*m, s, w*) gözetimli şekilde öğrenilir. Sonradan geliştirilen kod örneklerinden aynı ölçümler alınarak öğrenilmiş modele girdi olarak verilir ve modelin yaptığı tahmin çerçevesinde ilgili kodun tekrar tasarım gerektirip gerektirmediği belirlenir.

3 Deneyler ve Sonuçlar

3.1 Kullanılan Veriler

Tekrar tasarım tahmini probleminde önerilen modellerin sınanması için önde gelen bir GSM şirketinin altyapısını oluşturan yazılımların birbirini takip eden 4 versiyonu kullanılmıştır. İlgili yazılımlar Java ile geliştirilmiş orta katman uygulamalarıdır. Tüm versiyonlardan Halstead, McCabe ve satır sayısı ölçütleri toplanmıştır [12, 13, 14]. Toplanan ölçütlerin listesi Şekil 1'de ve kullanılan veriler hakkında genel bilgiler Tablo 1'de verilmiştir. Şekil 1'de listelenen kod ölçütleri Prest adını verdiğimiz ve araştırma laboratuvarımızda geliştirilen yazılım aracılığı ile elde edilmiştir [15]. Versiyonlar arası karşılaştırmalar sınıf ismi bazında yapıldığından, çalışmamızda yeniden isimlendirme tekrar tasarımları göz ardı edilmiştir.

Şekil 1'deki ölçütler kod sınıflarının karmaşıklıklarını ölçmeye yöneliktir. Tekrar tasarımın ardında yatan temel fikir karmaşıklığın azaltılması olduğundan, kullandığımız modellerde de buna yönelik girdiler kullanılması tercih edilmiştir.

Kullandığımız veri kümelerinde sınıfların tekrar tasarıma girip girmediğine dair bir bilgi bulunmamaktadır. Bu amaçla incelenen yazılımın 4 versiyonundaki tüm sınıflar için karmaşıklık ölçütleri çıkarılmış ve herbir versiyon için kendisinden sonra gelen versiyonlarda, ilgili sınıfın karmaşıklığının azalıp azalmadığı tespit edilmiştir. Sonraki versiyonlarda:

- Karmaşıklığı azalmış olan sınıflar tekrar tasarıma girmiştir ve
- Karmaşıklığı değişmeyen ya da artan sınıflar tekrar tasarıma girmemiştir

varsayımları yapılarak tüm sınıflar etiketlenmiş ve tekrar tasarım verisi bu varsayımlar sonucunda üretilmiştir.

Tablo 1. Çalışmada kullanılan 4 versiyonlu proje hakkında genel bilgi.

Versiyon	Ölçüt Sayısı	Sınıf Sayısı
2.19	26	524
2.20	26	528
2.21	26	528
2.22	26	534

CyclomaticDensity	HalsteadProgramDifficulty
DecisionDensity	HalsteadProgramLenght
EssentialDensity	HalsteadProgramLevel
BranchCount	HalsteadProgrammingEffort
ConditionCount	HalsteadProgrammingTime
CyclomaticComplexity	HalsteadProgramVolume
DecisionCount	MaintenanceSeverity
EssentialComplexity	CouplingBetweenObjects
Loc	FanIn
TotalOperands	NumberOfchildren
TotalOperators	PercentageOfPubData
UniqueOperandsNumber	ResponseForClass
UniqueOperatorsNumber	WeightedMethods

Şekil 1. Çalışmada kullanılan kod ölçütlerinin listesi. Anlaşılabilirlik açısından ölçüt isimleri İngilizce bırakılmıştır.

3.2 Performans Kriterleri

Tablo 2. Hata Matrisi

Gerçek	Tahmin Edilen	
	Tekrar Tasarım Var	Tekrar Tasarım Yok
Tekrar Tasarım Var	A	C
Tekrar Tasarım Yok	B	D

Sonuçlarımızı raporlarken, tekrar tasarımları doğru olarak yakalama olasılığı (pd) ve yanlış uyarı olasılığı (pf) olarak tanımlanan kriterleri kullandık [5, 6, 7, 10]. Yakalama olasılığı yalnızca gerçekten tekrar tasarımı yapılmış sınıflar içerisindeki başarımı ölçerken, yanlış uyarı olasılığı yalnızca tekrar tasarımı yapılmamış sınıflar içerisindeki kestirim hatasını ölçer. Bu performans kriterlerinin tanımları başarım ile birlikte, Tablo 2'de gösterilen hata matrisinin elemanları cinsinden Formül 4 ve Formül 5'te belirtilmiştir. Başarılı bir modelin pd değerinin yüksek ve pf değerinin düşük olması beklenmektedir.

$$pd = (A) / (A+C) \quad (4)$$

$$pf = (B) / (B+D) \quad (5)$$

3.3 Deney Tasarımı

Kullanılan modellerin genelleme yapabilme kapasitelerini ölçmek ve veri kümelerini ezberlemekten kaçınmak için bütün deneylerde 10 katlı çapraz geçerleme kullanılmıştır. Verilerin sırasının model performanslarını etkilemesini engellemek için çapraz geçerleme deneyleri, her seferinde verilerin sırası rastgele değiştirilerek 10 kez tekrarlanmıştır. Özet olarak her bir modelin her bir veri kümesi üzerindeki performansı 10x10=100 deneyin ortalama sonucu olarak raporlanmıştır [7]. Tüm gerçekleştirimler MATLAB ortamında standart araç takımları kullanılarak yapılmıştır.

3.4 Sonuçlar

Açıklanan modellerin aynı projenin birbirini takip eden 4 versiyonu üzerinde yaptığı tekrar tasarım tahminlerinin performansı Tablo 3'te özetlenmiştir. Projenin ilk versiyonunda, ağırlıklandırılmış Bayes modelinin tekrar tasarıma ihtiyaç duyan kod sınıflarının ortalama %63'ünü doğru tahmin ederken, sonraki versiyonlarda bu tahminlerin sırasıyla %90 ve %93 olduğu gözlemlenmiştir. Buradan çıkardığımız sonuç ise; modelimizin öğrenme performansının, daha kararlı versiyonlara ulaşıldıkça ve kodun karmaşıklığı hakkında daha fazla bilgi sahibi olundukça, arttığı şeklindedir. Ağırlıklandırılmış Bayes modelinin tekrar tasarıma ihtiyaç duyan kod sınıflarının ortalama %82'sini doğru tahmin ederken, %13 oranında yanlış uyarı verdiği gözlemlenmektedir. Girdileri eşit olarak ağırlıklandıran Naïve Bayes modelinde ise %76 oranında doğru tahmin ve %14 oranında yanlış uyarı gözlemlenmiştir. Önceki hata tahmini çalışmalarımızda da gözlemlediğimiz gibi ağırlıklandırma kullanan model standart modele gore daha üstün bir performans elde etmiştir. Gözlemlenen yüksek doğru tahmin oranları yazılım mimarları ve geliştiricilerine tekrar tasarım gerektiren sınıflar konusunda yol gösterici olurken, yalnızca %13 oranında yanlış uyarı elde edilmesi, önerilen modellerin pratikte kullanılabilirliğine işaret etmektedir.

Tablo 3. Tekrar tasarım tahmini sonuçları.

Versiyon	Naïve Bayes		Ağırlıklandırılmış Bayes	
	pd	pf	pd	Pf
2.19	49	17	63	16
2.20	88	13	90	12
2.21	92	12	93	13
Ortalama	***76***	***14***	***82***	***13***

4 Tartışma ve Gelecek Çalışma

Tekrar tasarım çalışmaları insan deneyimine dayalı, zaman alan ve maliyetli çalışmalardır. Bu konuda yardımcı araçların kullanılması hem insana bağımlılığı azaltacak hem de harcanan zaman ve maliyetleri düşürecektir.

Bu çalışmada tekrar tasarım problemi bir yapay öğrenme problemi olarak ele alınmıştır. Bu amaçla performansları hata tahmini çalışmalarında tatmin edici seviyede olan Naïve Bayes ve Ağırlıklandırılmış Bayes modelleri kullanılmıştır. Yazılım karmaşıklık ölçütlerinin girdi olarak kullanıldığı bu modellerin tekrar tasarım gerektiren sınıfların ortalama %82'sini doğru tahmin ettiğini ve yalnızca %13 oranında yanlış uyarı verdiği gözlemlenmiştir. Bu gözlemlerin yapıldığı deneyler yapay veri ya da küçük çaplı projeler üzerinde değil, önde gelen bir GSM şirketinin alt yapısını oluşturan yazılımlar üzerine gerçekleştirilmiştir.

Önerilen modellerin ürettiği sonuçların tekrar tasarım açısından yazılım mimarları ve geliştiricilerine yol gösterici ve zaman kazandırıcı olması beklenmektedir. Bu sayede değerli bilgi birikimine sahip bu insanlar daha verimli şekilde çalışabileceklerdir.

Gelecek çalışmamız aynı deneylerin başka projeler üzerinde, gerçek tekrar tasarım bilgileri toplanarak ve yeniden isimlendirme tekrar tasarımları da göz önüne alınarak tekrarlanması yönünde olacaktır. Ayrıca yapılan yanlış uyarıların azaltılması konusunda da modellerin iyileştirilmesi üzerine çalışılacaktır.

Kaynakça

1. Lehman, M.M. ve Belady, L.A., 1985. *Program evolution: processes of software change*, Academic Press Professional.
2. Fowler, M., Beck, K., Brant,J., Opdyke,W. ve Roberts,D., 2001. *Refactoring: Improving the Design of Existing Code*, Addison-Wesley.
3. Zhao, L. ve Hayes,J.H., *Predicting Classes in Need of Refactoring: An Application of Static Metrics.* In Proceedings of the Workshop on Predictive Models of Software Engineering (PROMISE), associated with ICSM 2006.
4. Simon, F. S. ve F., Lewerentz, C., *Metrics based refactoring*. Proc. European Conf. Software Maintenance and Reengineering, 2001.
5. Turhan, B. ve Bener, A., *Software Defect Prediction: Heuristics for Weighted Naive Bayes,* Barcelona, Spain, *ICSOFT 2007*
6. Turhan, B. ve Bener, A., *A Multivariate Analysis of Static Code Attributes for Defect Prediction*, Portland, USA, QSIC 2007
7. B. Turhan ve A. Bener, *Yazılım Hata Kestirimi için Kaynak Kod Ölçütlerine Dayalı Bayes Sınıflandırması*, UYMS 2007, Ankara, Türkiye, 2007
8. Ferreira, J.T.A.S., Denison, D.G.T. ve Hand, D.J., *Weighted naive Bayes modelling for data mining,* 2001
9. Lewis, D., *Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval.* Proceedings of ECML-98, 10th European Conference on Machine Learning, 1998
10. Menzies, T., Greenwald, J. ve Frank, A., *Data mining static code attributes to learn defect predictors.* IEEE Transactions on Software Engineering, 2007
11. Alpaydin, E., *Introduction to Machine Learning,* MIT Press, 2004
12. Welker, K. ve Oman, P.W., *Software maintainability metrics models in practice.* Journal of Defense Software Engineering, 1995

13. Hayes, J. ve Zhao., L,. *Maintainability Prediction: A Regression Analysis of Measures of Evolving Systems.* IEEE 21th International Conference on Software Maintenance, 2005

14. Halstead, M.H.ö *Elements of Software Science*, Operating, and Programming Systems Series, 1977

15. Turhan, B., Oral, A.D. ve Bener, A, *Prest- A tool for pre-test defect prediction.* Teknik Rapor, Boğaziçi Üniversitesi, 2007.

İlgilerin Birimsel Doğrulanması Üzerine Alan Araştırması[1]

Mustafa İspir[1], Aysu Betin Can[2]

[1,2] Enformatik Enstitüsü, Orta Doğu Teknik Üniversitesi, İnönü Bulvarı 06531, Ankara
[1] mustafa.ispir@synopsys.com, [2] aysu@ii.metu.edu.tr

Özet. Bu çalışmada ilgi yönelimli programlar (ing. aspect oriented programming) için ilgi sınıflaması, ilgi doğrulanması ve doğrulamaya yönelik tasarım hakkında alan araştırması yapılmıştır. İlgiler, sundukları güçlü özellikler sayesinde nesne yönelimli tasarımda birden fazla birime yayılan çapraz-kesen kavramların birimsel olarak gerçeklenmesini sağlamaktadır. Fakat ilgilerin bu güçlü özellikleri temel programın davranışını baştan sona değiştirebilmektedir. Bu durum yazılımın doğrulanmasını güçleştirmekte, birimsel akıl yürütmeyi zorlaştırmakta ve de en önemlisi ilgilerin yeniden kullanılmasını engellemektedir. Sayılan problemleri aşmak için ilgi yönelimli programlama, birimsel doğrulama yöntemleri ve araçları ile desteklenmelidir. Biz ilgi yönelimli proglamanın geleceğinde bu adımın önemli olduğunu düşünmekteyiz. Sunulan çalışma bu konuda yapılacak araştırmalar için bir başlangıç noktası olmak için yapılmıştır.

1 Giriş

Nesne yönelimli programlamanın soyutlama, sarmalama ve çok şekillilik özellikleri daha iyi uyumlu ve daha az bağımlı tasarımlar gerçekleştirmeyi amaçlamaktadır. Bu güçlü özelliklere rağmen iyi uyumlu ve az bağımlı tasarımlar gerçekleştirebilmek hala önemli bir sorundur. Bu zorluğun önemli nedenlerinden biri belli bir bileşene ait olmayan özelliklerdir. Birçok yazılım için özellikleri ikiye ayırabiliriz. Bir grup ozellik belli bir bileşende gerçeklenenleri temsil ederken, diğer grup birden fazla bileşende gerçeklenen özelliklerden oluşturur. Bu özelliklere çapraz-kesen özellikler (ing. cross-cuting concern) denir. Çapraz-kesen kavramlara örnek olarak günlük tutma, güvenlik politikası kontrolü, verilerin eşgüdümlü kullanımı gibi konuları verebiliriz. İlgi yönelimli programlama (ing. Aspect-oriented programming, AOP) çapraz kesen özelliklerin gerçekleştirimini kolaylaştırmak için geliştirilmiştir [1].

İlgiler (ing. aspect) ile gerçeklenmiş birimler sistemin geri kalanına dokunarak (ing. weaving) entegre olur ve çalışan sistemi oluşturur. İlginin dokunduğu programa bundan sonra temel program denecektir. Oluşan son sisteme ise dokunmuş sistem diyeceğiz. İlgilerin temel programda dokunabileceği noktalara katılım noktası (ing. joinpoint) denir. Fonksiyon çağırma, fonksiyon yürütme, alan güncelleme ve alana ulaşım örnek katılım noktalarıdır. Katılım noktalarından boole verilerine tanımlı fonksiyona kesim noktası (ing. pointcut) tanımı denir. Kesim noktaları ise bu tanımın doğru değer verdiği kesim noktalarıdır. İlgiler içinde önerilerde (ing. advice) kodlanmış program, temel programın kesim noktalarında aktif olur.

[1] Bu çalışma TÜBİTAK 106E032 numaralı proje tarafından desteklenmektedir.

Temel program ilgiler hakkında herhangi bir bilgiye sahip değildir. Bu yüzden temel program üzerinde iş yapan bir yazılım mühendisinin, ilgilerden kaynaklanacak etkileri görmesi kolay bir iş değildir. Temel programın ilgiden habersizliğinin yanı sıra, ilgilerin çok güçlü özellikleri olması birimsel akıl yürütmeyi oldukça zorlaştırmaktadır. İlgiler bütün temel programın akışını değiştirebilir. Bu yüzden ilgi kodlarındaki değişikliklerin doğrulanması çok önemlidir. Her değişiklikten sonra bütün sistemi ilgilerle dokuyup doğrulamak ise pek uygulanabilir bir yöntem değildir. Bu nedenlerle ilgi yönelimli programlamanın birimsel doğrulama yöntemleri ve araçları ile desteklenmesi gerektiğini düşünüyoruz. Ayrıca ilgilerin yeniden kullanımı ilginin belli bir temel kodtan bağımsız olarak birimsel doğrulanmasını gerektirmektedir.

Dokunmuş bir programda sağlanması beklenen özellikleri birimsel olarak analiz yapabilmek için iki gruba ayırmaktayız. Bunlar *temel özellikleri* ve *ilgi özellikleridir*. Temel özellikleri, temel programın ilgilerden bir katkı almadan sağladığı özelliklerdir. Bu özelliklerin doğrulanmasında cevabı aranacak soru şudur: "ilgi dokunduktan sonra temel özellikleri korunuyor mu?" İlgiler tarafından temel programın bir katkısı olmadan sağlanan özelliklere ise ilgi özellikleri diyoruz. İlgi özelliklerinin sağlanması için temel programdan beklenen, ilgi önerilerini uygun bir şekilde aktive etmesidir. Yani temel programın ilginin varsaydığı durumları sağlaması gerekmektedir. İlgi özelliklerinin doğrulanması sırasında cevabı aranacak iki soru vardır. Birincisi "İlgi, kendinden beklenen özelliği sağlamakta mıdır?" İkincisi ise "Temel program ilginin varsayımlarıyla uyumlu mudur?"

Gerek varsayımların güçlü bir şekilde ifade edilebilmesine gerekse de otomatik doğrulamaya izin verdiği için model denetlemenin bu alanda katkısı olacağını düşünmekteyiz. Model denetlemesinden bahsettikten sonra bu konuda yapılmış çalışmalara değineceğiz. Alan taraması üç ayrı başlık altına bölünmüştür. Bu başlıklar ilgilerin sınıflandırılması, ilgi doğrulanması ve doğrulamaya yönelik tasarımdır. İlgi sınıflamasının ilgileri doğrulamak için önemli ipuçları içerdiğini düşünmekteyiz. Doğrulamaya yönelik tasarım yöntemleri ise doğrulama ile tasarım arasındaki sinerjiden faydalanmaya çalışmaktadır. Son olarak, sonuç kısmında bu alanda yapılacak bir çalışmanın dikkate alması gereken noktaları belirteceğiz.

2 Model Denetleme

Model denetleme [2] bir otomatik formal doğrulama tekniğidir. Model denetleme, verilen bir modelin istenen bir özelliği sağlayıp sağlamadığını kontrol eder. Formal doğrulama, özellikle kritik sistemlerin hatasız olduğunu göstermek için uygulanmaktadır. Benzetim ve sınama yöntemleriyle doğrulama bu sistemler için yeteri kadar güvenli değildir. Bunun nedeni benzetim ve sınama yöntemlerinin bir sistemin verili girdiler için doğru çalıştığını göstermesine rağmen sistemin başka girdilerle yanlış çalışmadığını iddia edememesidir. Model denetleme, sistemin bütün koşullar altında verili özellikleri sağlayıp sağlamadığına cevap vermektedir. Model denetlemenin diğer formal doğrulama yöntemlerine karşı en önemli avantajı otomatik doğrulama yapabilmesidir. Model denetleme süreci üç ana adımdan oluşmaktadır. Bunlar modelleme, doğrulanacak özelliklerin belirlenmesi ve doğrulamadır.

Model denetleme algoritmalarının kullandığı model Kripke yapısıdır. Kripke yapısı bir sonlu durum makinesidir. Durumların etiketleri vardır. Bu etiketler sistemde tanımlı Boole ifadelerininden o durum içinde doğru değeri alanlarını gösterir. Bu ifadelere atomik özellikler denir. Algoritmaların Kripke yapısı üzerinde tanımlanmış olmasına rağmen modelleme pratikte model denetleme araçlarının girdi olarak kullandığı dillerde yapılır. Örnek olarak eğer model denetleme aracı olarak SPIN [3] kullanılıyorsa sistem Promela diliyle modellenir. Modelleme sırasında en büyük zorluk durum uzayının çok hızlı büyümesi ve patlamasından kaynaklanır. Durum uzayını küçültmek için gerçek sistem üzerinde soyutlama teknikleri kullanılarak model oluşturulur. Soyutlamanın model denetlemeyi uygulanabilir yaparken doğrulama sonuçlarını tehlikeye atmaması gerekmektedir.

Doğrulanacak özellikleri belirtmenin popüler yolu zamansal mantıktır. Zamansal mantık doğruluğu zamana bağlı olan önermeleri tanımlamak için kullanılır. Örnek olarak "uyuyana kadar başım ağrıyacak" önermesi zamansal mantık ile belirtilebilir. Zamansal mantık Pnueli'den [4] sonra bilgisayar bilimi alanında programların belirtilmesi ve doğrulanması için sıkça kullanılmaya başlanmıştır. Linear Temporal Logic (LTL) ile Computation Tree Logic (CTL) model denetleme alanında en çok kullanılan özellik belirtim mantıklarıdır. Belirtilen özelliklerin doğruluğu ve eksizlikliğinden bunları tanımlayan kişi sorumludur.

Model denetlemenin son aşaması verilen modelin verili özellikleri sağlayıp sağlamadığını denetlemektir. Bu aşama araçlar tarafından otomatik olarak yapılmaktadır. Modelin özellikleri sağlamadığı durumda, araçlar hatanın oluştuğu örnek bir yürütme izini raporlarlar. Böylelikle sadece hatanın olup olmadığını bilmekle kalmaz, hatayı nasıl oluşturabileceğimizi de öğreniriz. Java PathFinder (JPF) [5] Java programları için tasarlanmış bir model denetleme aracıdır. Byte kod seviyesinde çalışır. En büyük avantajı modelleme dilinin Java olmasıdır. Böylelikle modellemeye iş gücü ayrılmamaktadır. Bu avantajın bedeli denetlenebilecek programların boyutlarının küçülmesidir. Bir diğer araç SPIN'dir. SPIN'in modelleme dili Promela'dır. Doğrulanacak özellikler LTL ile belirtilir. SPIN'in odaklandığı problem kümesi paralel çalışan iş parçacıkları arasındaki etkileşimdir.

3 İlgilerin Sınıflandırılması

İlgilerin doğrulanmasında kullanışlı olacağını düşündüğümüz iki gruplama yöntemini seçtik. Bu yöntemlerden ilki Rinard [6] tarafından önerilmiştir. Rinard'ın gruplama yöntemi öneri kodu ile temel kod arasındaki veri alışverişinin ve kontrol akışının özelliklerine dayanmaktadır. Tanıtacağımız ikinci yöntem Katz [7] tarafından önerilmiştir. Katz'ın gruplandırma yöntemi, temel programın sonlu durum makinesi ile dokunmuş programın sonlu durum makinesinin arasındaki farklara dayanmaktadır.

Rinard'ın yöntemi önerileri iki boyutlu şekilde gruplamaktadır. Boyutlardan biri öneri kodu ile temel kod arasında veri alışverişine, diğeri ise kotrol akışına dayanmaktadır. Bu gruplamayı otomatik şekilde yapan bir yöntem de önerilmiştir. Aşağıdaki tanımlarda geçen temel koddan kasıt önerinin aktif olduğu kesim noktasının kapsamında kalan koddur. Kontrol akışına dayalı boyutta dört farklı grup tanımlanmıştır:

- *Genişletme Önerisi (ing. Augmentation Advice)*: Büyütme önerisi temel koda dokunduktan sonra, temel kodun tamamı yürütülmeye devam eder.
- *Daraltma Önerisi (ing. Narrowing Advice)*: Daraltma önerisi temel koda dokunduktan sonra, ya temel kodun tamamı yürütülür ya da hiç yürütülmez.
- *Değiştirme Önerisi (ing. Replacement Advice)*: Değiştirme Önerisi dokunduktan sonra, temel kod yürütülmez.
- *Birleşim Önerisi (ing. Combination Advice)*: Eğer öneri yukarıdaki gruplardan hiç birine girmiyorsa birleşim önerisidir.

Veri alışverişine dayalı boyutta ise beş grup vardır. Bu gruplara önerinin kapsamı denir. Önerinin kapsamı aşağıdakilerden biri olabilir:

- *Dikey (ing. Orthogonal)*: Temel kod ile önerinin ortak okuduğu ya da yazdığı bir alan yoktur.
- *Bağımsız (ing. Independent)*: Temel kodun yazdığı bir alan, öneri tarafından okunmaz. Aynı şekilde öneri tarafından yazılan bir alan temel kod tarafından okunmaz. Buna karşın iki kodda aynı alanı okuyabilir.
- *Gözetleme (ing. Observation)*: Gözetleme kapsamının bağımsız kapsamdan tek farkı, öneri kodunun temel kod tarafından yazılan alanları okuyabilmesidir.
- *Hareketlendirme (ing. Actuation)*: Hareketlendirme kapsamının bağımsız kapsamdan tek farkı, temel kodun öneri kodu tarafından yazılan alanları okuyabilmesidir.
- *Girişim (ing. Interference)*: Girişim kapsamında öneri ve temel kod aynı alanı değiştirebilirler.

Rinard'a göre genişletme önerisinin dikey, bağımsız ve gözetleme kapsamları kullanılması birimsel akıl yürütmeye izin verir. Birimsel akıl yürütme şu anlamda kullanılmıştır: temel kodu yazan kişi o kod üzerinde sonuçlara varırken öneri kodu ile ilgilenmek zorunda kalmaz; aynı şekilde öneri kodu üzerinde çalışan kişi o kod üzerinde sonuçlara varırken temel kod ile ilgilenmek zorunda kalmaz. Ayrıca önerilerin tiplerini ve kapsamlarını bilmenin de yazılım mühendisini akıl yürütme sırasında olumlu yönlendireceği belirtilmiştir.

Katz ilgileri, ilginin temel kodun Kripke yapısı üzerinde yaptığı değişikliğe göre gruplamaktadır. Bu gruplama biçiminin motivasyonu verili bir zamansal mantığın geçerliliğini denetlerken en basit yöntemi uygulayabilmektir. Yani ilginin hangi grupta olduğuna bakılarak denetleme sırasında olabilecek en basit yöntem seçilir. Bu açıdan Katz kendi gruplaması ile zamansal özelliklerin gruplaması arasındaki ilişkiyi incelemiştir. Zamansal özellikler güvenlik, canlılık ve varlık olarak gruplandırılır [8]. Katz önerdiği gruplamaların statik kod analizi ile nasıl bulunacağını da belitmiştir. Önerilen ilgi gruplaması ve bunların zamansal özelliklerle ilgisi aşağıda verilmiştir:

- *Seyirci İlgi (ing. Spectative Aspect)*: Seyirci ilgi ile dokunmuş kodun Kripke yapısı ile temel kodun Kripke yapısı bazı durumların tekrarlanmış olması ve fazladan atomik özellikler gelmesi dışında aynıdır. Bu grup Rinard'ın

gruplmasında genişletme önerisinin dikey, bağımsız ve gözetleme kapsamlarına denk düşmektedir. Zamansal özelliklerden *sonraki durum* (X operatörü ile gösterilir) dışında temel kodda geçerli olan herhangi bir özellik, dokunmuş kodda da geçerlidir. İlgi kodu üzerinde geçerli sabit özellikler, dokunmuş kod üzerinde de geçerlidir.

- *Düzenleyici İlgi (ing. Regulative Aspect)*: Düzenleyici ilgi ile dokunmuş kodun Kripke yapısı ile temel kodun Kripke yapısı bazı durumların tekrarlanmış olması, fazladan atomik özellikler gelmesi, bazı durumların ve bağlantıların silinmiş olması dışında aynıdır. Bu grup Rinard'ın gruplamasında daraltma önerisinin dikey, bağımsız ve gözetleme kapsamlarının biraz daha kısıtlanmış halidir. Bu kısıtlama, eğer öneri temel kodun yürütülmesini engelliyorsa yürütmeyi iptal etmesi şeklindedir. Buna örnek olarak güvelik protokolü kotrolü ile ilgili bir öneriyi verebiliriz. Temel kod üzerinde geçerli bütün güvenlik özellikleri dokunmuş kod üzerinde de geçerlidir. Canlılık ve varlık özellikleri otomatik olarak korunmazlar. Bu özelliklerin tekrardan doğrulanması gerekmektedir.
- *İstilacı İlgi (ing. Invasive Aspect)*: Eğer öneri temel kodun Kripke yapısını yukarıdaki gruplara giremeyecek şekilde değiştirdiyse, istilacı ilgidir. Herhangi bir zamansal özelliğin korunup korunmadığı bilinemez. Tekrar denetleme gereklidir.
- *Zayıf İstilacı İlgi (ing. Weakly Invasive Aspect)*: Zayıf istilacı ilgi, istilacı ilginin bir altkümesidir. Dokunmuş kodun Kripke yapısında, ilgi kodundan çıkış temel kodda eskiden ulaşılan durumlardan birine olursa bu ilgiye zayıf istilacı ilgi denir. Bu durumda temel kodda geçerli olan sabit özelliklerin denetiminde kolay bir yöntem uygulanabilir. Eğer ilgi kodunda bu özellik girişte özellik korunuyordur varsayımıyla denetlendiğinde geçerli ise dokunmuş kodda da geçerlidir.

4 İlgilerin Doğrulanması

Bu bölümde ilgilerin doğrulanması konusunda yapılmış çalışmaları aktarıyoruz. Özellikle birimsel doğrulama ya da model denetleme yöntemlerini kullanan çalışmalara yer verdik. Birimsel doğrulama yöntemlerinin ilgilerde kullanılabileceği ilk olarak Devereux [9] ve Sipma [10] tarafından ortaya koyulmuştur. Temelde katılım noktaları üzerindeki varsayımların sağlandığı durumda, önerilerin sonunda belli özelliklerin sağlanıp sağlanmadığına bakılmaktadır.

Denaro ve diğ. [11], bir deney olarak bir senkronizasyon ilgisinin model denetlemesini yapmışlardır. Bu ilginin Promela modeli oluşturulup SPIN ile denetleyerek bir kilitlenmeye neden olmadığı gösterilmiştir. Bu çalışmada ilginin temel programa dair varsayımlarının nasıl kullanılacağı gösterilmemiştir. Ayrıca modelin elle yazılması hem zaman alıcı hem de model denetleme konusunda uzman yazılım mühendisleri gerektirmektedir.

Goldman ve diğ. [12] ilgi özelliklerini doğrulamak için umut vaat edici bir yöntem önermiştir. Bu yöntemde temel program hakkındaki varsayımlar LTL ile belirtilmiştir. LTL özelliklerini sağlayan en genel Kripke yapısını oluşturan algoritma [2] kullanılarak temel programın en genel hali modellenmiştir. Kesim noktalarının tanımları kulalnılarak

temel programın modeli üzerinde kesim noktalarının aktive olduğu durumlar bulunmuştur. Bu bilgiler kullanılarak dokunmuş programın modeli elde edilmiştir. İlgi özellikleri, dokunmuş programın modeli üzerinde denetlenmiştir. Temel programın ilginin varsayımları ile uyumlu olduğunu göstermek için LTL özelliklerini sağlayıp sağlamadığına bakılır. Bu yöntem de model denetleme konusunda uzman yazılım mühendisleri gerektirmektedir. İstilacı ilgiler bu yöntemle denetlenememektedir. Son olarak sadece bir ilgi ve bir öneriden oluşan bir program denetlenmiştir.

Krishnamurthi ve diğ. [13], temel özellikler için birimsel doğrulama yöntemi önermişlerdir. "İlgi dokunduktan sonra temel programın özellikleri korunur mu?" sorusuna cevap vermişlerdir. Bu çalışmanın ana katkısı temel programın Kripke yapısı üzerinde çalışan ve kesim noktalarının aktive olduğu durumları bulan bir algoritmadır. İlgilerin temel program özellikleri ile çelişip çelişmediği, önerilerin aktive olduğunda temel programda olan özellikleri nasıl değiştirdiği ve bu değişikliğin önerinin çıkış noktasındaki özellikleri sağlayıp sağlamadığına bakılarak anlaşılır. Bu yöntem ile ilgide yapılacak değişiklikler hızlıca denetlenebilmektedir. Bu yöntemin kullanılmasını güçleştiren iki engel vardır. Birincisi temel programdaki her değişiklikten sonra bütün sürecin tekrarlanması gerekmektedir. İkincisi ise bu yöntem sadece fonksiyon çağırmalarına tanımlanan kesim noktalarını incelemektedir.

Ubayashi ve diğ. [14], ilgiler içinde bulunan doğru çalışma gözcülerini (ing. assertion) ayrı bir ilgiye toplamayı önermişlerdir. Bu sayede ilgi kodu ile denetleme kodu birbirinden ayrılacaktır. Dokunmuş program ise JPF ile denetlenerek herhangi bir hata olup olmadığına bakılır. Önerilen yöntem birimsel doğrulama olmayıp, denetleme kodunun birimselleştirilmesidir. Bu açıdan durum uzayı patlaması problemine dair bir katkıda bulunmamışlardır. Bu yöntem JPF'i dokunmuş bütün sistem üzerinde kullandığı için uygulanabileceği proje boyutu sınırlıdır.

Dantas ve diğ. [15] sadece zararsız önerilere izin veren ilgi yönelimli programlama dili önermişlerdir. Zararsız öneriler ya program akışına karışmamakta ya da programdan çıkış sağlamaktadırlar. Güvenlik politikalarını denetleyen ilgiler bu kategoriye iyi bir örnektir. Zararsız öneri tanımı AspectJ ile gerçeklenebilecek önerilerin küçük bir alt kümesini oluşturmaktadır. Buna rağmen, önemli ölçüdeki ilgi kullanımının zararsız öneri kapsamına girdiğini idda etmektedirler. Rinard'ın sınıflamasına göre zararsız öneriler daratlma ve genişletme önerilerinin dikey, bağımsız ve gözetleme kapsamlarına girmektedirler. Katz'ın sınıflamasında ise düzenleyici ve seyirci ilgi sınıfına aittirler. Zararsız öneriler sayesinde temel programı yazan kişinin ilgiden bağımsız olarak birimsel akıl yürütebileceğini söylemektedirler. Katz'ın da belirttiği gibi düzenleyici ilgiler canlılık ve varlık özelliklerin korunmasını garanti etmemektedirler. Bu yüzden idda edilenin aksine bu özellikler için dokunmuş program denetlenmelidir.

Nelson ve diğ. [16] ilgi modelleri kullanılarak dokunmuş programın modelini oluşturan bir algoritma önermektedirler. Bunu ilgi derleyicilerinin dokuma algoritmasını modeller için kullanmak olarak düşünebiliriz. Örnek olarak etiket geçiş sistemi [17] (ing. label transition system) ile modellenmiş ilgiler üzerinde algoritma denenmiştir. Önerilen yöntem birimsel akıl yürütmeyi desteklemediği gibi durum uzayı patlama problemine bir çözüm sunmamaktadır.

5 Doğrulamaya Yönelik Tasarım

Model denetleme otomatik bir doğrulama yöntemi olsa da endüstri projelerinde uygulamak çok fazla emek gerektirmektedir. Çalışmanın büyük kısmı tasarımı birimselleştirmeye ve uygun bir seviyeye soyutlamaya gitmektedir. Bu işlemler sayesinde yazılım model denetleme algoritmaları uygulanabilir bir seviyede ifade edilir. Bu işlemler model denetleme konusunda uzman mühendisler gerektirmektedir. Birimselleştirmenin ve soyutlamanın tasarım ekibinin de hedefleri arasında yer alması tasarım ve doğrulama arasında işbirliği olabileceğini düşündürmüştür. Benzer bir şekilde, test ile sürülen geliştirme (ing. test drivien development) araştırmaları test edilebilirlik çabasının kaliteli tasarımlara yönlendirdiğini göstermiştir [18]. Doğrulamaya yönelik tasarım (ing. design for verification, D4V) [19], tasarım sürecinde doğrulamaya yönelik bilgilerin de içerilmesi gerektiğini belirtir. Örnek olarak kontrat ile tasarım (ing. design by contract) [20] bir çeşit doğrulamaya yönelik tasarım yöntemi olarak düşünülebilir. D4V özellikle donanım tasarımı için de güncel bir konudur [21] [22] [23] [24]. İlgi yönelimli programlamaya uygun D4V yöntemlerinin geliştirilmesi önemli bir katkı sağlayacaktır.

Bultan ve diğ. [19] D4V yöntemlerinin aşağıdaki soruların bazılarına cevap verdiklerini belirtmişlerdir. Bu soruları D4V araştırmalarını yönlendiren sorular olarak düşünebiliriz.

- Tasarım aşamasındaki birimselleştirme, soyutlama ve arıtma işlemleri doğrulama yöntemleriyle daha iyi nasıl bağlanabilir?
- Sonlu durum makineleri doğrulama yöntemlerine katkı sunmak amacı ile tasarım aşamasında nasıl etkili kullanılabilirler?
- Arayüzler nasıl tanımlanmalı ki doğrulama aşamasında kullanılabilsinler?
- Tasarım ve gerçekleme dilleri nasıl değiştirilmeli ki otomatik doğrulama yöntemlerinin verimliliği artsın?
- Doğrulanabilir tasarımlar üretmek için şu anda kullanılan tasarım yöntemleri evrimsel olarak iyileştirilebilir mi yoksa devrimsel bir şekilde yeni bir tasarım yöntemi mi önerilmeli?

Betin-Can ve diğ. [25] [26] [27] ile Mehlitz ve diğ. [28] otomatik doğrulama yöntemlerini mümkün kılacak tasarım desenleri önermektedirler. Mehlitz ve diğ. [29] tasarım desenleri formatını genişleterek tasarım kontratı bilgisini içermesini sağlamaktadır. Bu yöntem tasarım desenlerinin doğrulanmasını kolaylaştırmasına rağmen otomatik doğrulamanın uygulanabilir olduğu proje büyüklüğüne katkıda bulunmamaktadır. Betin-Can ve diğ. sonlu durum makinesini arayüz tanımını ayrıntılandırmak için kullanmaktadırlar. Önerilen yöntem ile arayüzde sunulan yöntemlerin çağrılma sırası ile ilgili kural tanımlanabilmektedir. Tasarım desenlerinin genel kullanımı ile otomatik doğrulama yöntemlerinin formal tanımlama ihtiyacı arasında bir çelişki vardır. Bu çelişkiyi aşmak için alana özel tasarım desenleri önerilmektedir. Betin-Can yaklaşımını koşutzamanlı programların denetleme işlemleri düzenleyecek bir tasarım kalabı ve ağ hizmetleri (ing. web service) arasındaki iletişime odaklanan bir tasarım kalıbı önermiştir. Betin-Can tarafından önerilen tasarım kalıpları

ve yöntem birimsel doğrulamayı sağladığı için otomatik doğrulamanın kullanılabileceği proje büyüklüğüne arttırmaktadır.

Giannakopoulou [30] ve diğ, otomatik birimsel model denetlemeyi mümkün kılacak doğrulamaya yönelik tasarım yöntemi önermişlerdir. Tasarım sürecinde birimlerin davranışsal arayüzünü sonlu durum makineleri ile tanımlamışlardır. Bu arayüzler sayesinde birimlerin kullanım varsayımları ve kullanıcılara karşı garanti ettiği sonuçlar tanımlanabilmektedir. Bu bilgiler kullanılarak model denetleme yöntemi otomatik olarak kullanılabilmektedir. Önerilen yöntem NASA'nın gezgin robotu K9'un yönetim altbiriminin geliştirilmesinde kullanılmıştır.

6 Sonuç

İlgi yönelimli programlama heyecan uyandırmış olmasına rağmen endüstri projelerinde beklenen yaygınlıkta kullanılmamaktadır. Bunun nedeni olarak nesne yönelimli programlamada bulunan yöntem ve araç olgunluğunun ilgi yönelimli programlamada olmamasını görmekteyiz. Örnek olarak, nesnelerin tekrar kullanımı ve birimsel doğrulanması konusunda olgunlaşmış araçlar ve yöntemler vardır. İlgilerin birimsel doğrulanması ve tekrar kullanımı üzerine çalışmalar ise devam etmektedir. Model denetlemenin bu alanda katkısı olacağını düşünmekteyiz. Model denetlemenin son zamanlarda gündemde olmasının bir nedeni de algoritmalardaki ve bilgisayar işlem gücündeki hızlanmalardır.

Bu konuda yapılacak çalışmalara yön gösterebilmesi için önemli noktaların altını çizmek istiyoruz. Geliştirilecek yöntemin yazılım mühendisleri tarafından kullanılabilmesi gerekmektedir. Bu açıdan yöntemin kullanılan programlama dilini kısıtlamaması ve ileri derecede uzmanlaşmış yazılım mühendislerine ihtiyaç duymaması avantaj sağlar. Yöntemin endüstri projelerinde uygulanabilir olması için büyük ölçekli projelerde çalışabilir olmasını beklemekteyiz. Önerilecek yöntemin yaygın bir şekilde kullanılması için de yazılım geliştirme sürecine kolaylıkla bütünleşmesi gerekmektedir. Şu ana kadar yapılmış çalışmalar saydığımız özellikleri karşılamaktan uzaktır.

Kaynakça

1. Elrad, T., Filman, R. E., Bader, A. (2001) . Aspect Oriented Programming: Introduction. In Commun. ACM, 44(10):29-32.
2. Clarke, E. M., Grumberg, O., ve Peled, D. A. (2000). Model Checking. The MIT Pres.
3. Holzmann, G. J. (1997). The model checker spin. Software Engineering, 23(5):279-295.
4. Pnueli, A. "The Temporal Logic of Programs", 19th Annual Symposium on Foundations of Computer Science, Providence R,I. Nov. 1977.
5. Visser, W., Havelund, K., Brat, G., Park, S., ve Lerda, F. (2003). Model checking programs. Automated Software Engineering, 10(2):203-232.
6. Rinard, M., Salcianu, A., ve Bugrara, S. (2004). A classification system and analysis for aspect-oriented programs. In SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering, s. 147-158, New York, NY, USA. ACM Press.

7. Katz, S. (2006). Aspect Categories and Classes of Temporal Properties. In Transactions on Aspect Oriented Software Development, Volume 1, LNCS 3880: 106-134.

8. Manna, Z., Pnueli, A. (1991). The temporal logic of reactive and concurrent systems: specification, Srpinger, Berlin Heidelberg New York.

9. B. Devereux. Compositional reasoning about aspects using alternating-time logic. In Proc. of Foundations of Aspect Languages Workshop (FOAL03), 2003.

10. H. Sipma. A formal model for cross-cutting modular transition systems. In Proc. of Foundations of Aspect Languages Workshop (FOAL03), 2003.

11. Denaro, G., Monga, M. An Experience on Verification of Aspect Properties. Proceeding of the International Workshop o Principles of Software Evolution (IWPSE), Vienna, Austria, Sept. 2001.

12. M. Goldman ve S. Katz. Modular generic verification of LTL properties for aspects. In Foundations of Aspect-Oriented Languages (FOAL), Mar. 2006.

13. Krishnamurthi, S., Fisler, K., ve Greenberg, M. (2004). Verifying aspect advice modularly. In SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering, volume 29, s. 137-146, New York, NY, USA. ACM Press.

14. Ubayashi, N. ve Tamai, T. (2002). Aspect-oriented programming with model checking. In AOSD '02: Proceedings of the 1st international conference on Aspect-oriented software development, s. 148-154, New York, NY, USA. ACM Press.

15. Dantas, D. S. ve Walker, D. (2006). Harmless advice. In POPL '06: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, s. 383-396, New York, NY, USA. ACM Press.

16. Nelson, T., Cowan, D. D., ve Alencar, P. S. C. (2001). Supporting formal verification of crosscutting concerns. In REFLECTION '01: Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, s. 153-169, London, UK. Springer-Verlag.

17. Magee, J., Kramer, J. (1999) Concurrency: State Models and Java Programs. John Wiley & Sons.

18. Beck, K. (2002). Test Driven Development: By Example. Addison-Wesley Professional.

19. Bultan, T.; Heitmeyer, C.; O'Leary, J., "Panel on design for veri~cation," Formal Methods and Models for Co-Design, 2005. MEMOCODE '05. Proceedings. Third ACM and IEEE International Conference on , s. 232-235, 11-14 Tem. 2005.

20. Jazequel, J.-M.; Meyer, B., "Design by contract: the lessons of Ariane," Computer , vol.30, no.1, s.129-130, Haz. 1997.

21. Moorby, P., "Design for verification with System Verilog," VLSI Design, 2004. Proceedings. 17th International Conference on, 2004.

22. Malay K Ganai; Akira Mukaiyama; Aarti Gupta; Kazutoshi Wakabayshi, "Synthesizing "Verification Aware" Models: Why and How?," VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on, s.50-56, Haz. 2007.

23. Bombieri, N.; Fummi, F.; Pravadelli, G., "A TLM Design for Verification Methodology," Research in Microelectronics and Electronics 2006, Ph. D. , s. 337-340, 12-15 Haz. 2006.

24. Haja Moinudeen; Ali Habibi; Sofiene Tahar, "Design for Verification of the PCI-X Bus," Formal Methods in Computer Aided Design, 2006. FMCAD '06 , s.187-188, Kas. 2006.

25. A. Betin-Can ve T. Bultan. Verifiable web services with hierarchical interfaces. In Proceedings of the 2005 IEEE International Conference onWeb Services (ICWS 2005), s. 85–94, 2005.

26. A. Betin-Can, T. Bultan, M. Lindvall, S. Topp, ve B. Lux. Eliminating synchronization faults in air traffic control software via design for verification with concurrency controllers. Automated Software Engineering Journal, 14(2):129–178, Haz. 2007.

27. A. Betin-Can. Design for Verification for Concurrent and Distributed Systems. PhD thesis, University of California Santa Barbara, 2005.

28. Mehlitz, P. C., Penix, J. Design for Verification using design patterns to build reliable systems. In Proc. Work. On Component-Based Soft. Eng., 2003.

29. Gamma, E., Helm, R., Johnson, R., ve Vlissides, J. (1995). Design Patterns. Addison-Wesley Professional.

30. D. Giannakopoulou, C. S. Pasareanu ve J. M. Cobleigh. Assume-Guarantee Verification of Source Code with Design-Level Assumptions. Proceedings of the 26 th International Conference on Software Engineering, vol. 23, no. 28, s. 211, 2004.

Hareket Eden Nesnelerin MADS Belirtim Dili ile Modellenmesi ve Gerçeklenmesi

Perihan Kilimci[1], Oya Kalıpsız[2]

[1] Hava Harp Okulu Komutanlığı, Bilgisayar Mühendisliği Bölümü, 34149, Yeşilyurt, İstanbul
[2] Yıldız Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü, 34349, Beşiktaş, İstanbul
[1] p.kilimci@hho.edu.tr, [2] kalipsiz@yildiz.edu.tr

Özet. İnternet kullanımının ve kablosuz cihazların yaygınlaşması ile hem konuma hem de zamana bağlı (sapatio-temporal) sorgulamaların yapılmasına olanak sağlayan uygulamalar geliştirilmiştir. Uygulamaların bu şekilde çeşitlilik kazanmasına karşılık, hem konum hem de zaman boyutlarının eklenmesi ile oluşturulan veri modelleri karmaşıklaşmıştır. Veri modellerinin daha çabuk kavranabilmesi için konum ve zaman boyutu çeşitlerini gösteren çeşitli semboller kullanılarak veri modeli yeniden MADS (Modeling of Application Data with Spatio-temporal features) ile tanımlanmıştır. Bu çalışmada Christine Parent ve arkadaşları tarafından geliştirilmiş ve geçtiğimiz on yıl içinde uygulanmış, MADS modelleme yapısı incelenmiştir. Daha sonra, hem konuma hem de zamana bağlı özellikleri olan bir hareket eden nesneler uygulamasının modelleme ve uygulama çalışmaları verilmiştir. Veritabanında saklanan katmanlı Eminönü sayısal haritası bir grafik arayüzü ile görselleştirilmiştir. Son olarak, gerçekleştirilen konuma bağlı, zamana bağlı ve hem konuma hem de zamana bağlı çeşitli sorgular çalışmaya eklenmiştir.

1 Giriş

İnternet kullanımının ve internete bağlı uygulamaların artması ile, konuma bağlı (spatial) uygulamalar önem kazanmıştır. Örneğin sabit bir bilgisayardan internete bağlanan kullanıcılar, belirli bir yere 5 km. uzaklıkta bulunan restorantları veya hastaneleri sorgulayabilme gibi konuma bağlı çeşitli sorguları gerçekleştirebilmişlerdir. Bu uygulamalar, kablosuz cihazların ve cep telefonlarının kullanımının yaygınlaşması ile gelişmiş ve hem konuma hem de zamana bağlı (sapatio-temporal) sorgulamaların yapılmasına olanak sağlamıştır. Artık hareket etme yeteneğine sahip internet kullanıcıları, bulundukları yere 5 km. uzaklıkta bulunan ve yarım saat içinde erişebilecekleri restorantları veya hastaneleri sorgulama gibi hem konuma hem de zamana bağlı çeşitli sorgulamaları yapabilmektedir.

Uygulamaların bu şekilde çeşitlilik kazanmasına karşılık veri modelleri, geleneksel veritabanı sistemlerinde varlık ilişki modelleri (Entity-Relationship) veya nesneye yönelik programlarla oluşturulan sistemlerde UML (Unified Modeling Language) ile gerçekleştirilmeye çalışılmıştır. Fakat, uygulamalara hem konum hem de zaman boyutlarının eklenmesi ile oluşturulan modeller karmaşıklaşmıştır. Veri modellerinin daha çabuk kavranabilmesi için konum ve zaman boyutu çeşitlerini gösteren çeşitli semboller kullanılarak veri modeli yeniden MADS (Modeling of Application Data with Spatio-temporal features) ile tanmlanmıştır[1].

Bir veri modeli, gerçek dünyadaki uygulamada bulunan kavramların bilgisayar sistemlerindeki gösterimidir. Algılanan dünya ile kavramların gösterimi arasında doğrudan bir eşleme varsa bu model, kavramsal model olarak adlandırılmaktadır. Kavramsal modellerin gücü, kavramlarının sayısı ve bunlar arasındaki ilişkiler yaratabilme yeteneği ile ilişkilidir. Veri modeli kullanıcıları, oluşturulan kavramsal modelin çok karmaşık olmamasını ve kolay kullanılmasını talep etmektedir. Bu nedenlerle de oluşturulan kavramların görsel gösterimi de açık olarak kullanıcı tarafından anlaşılabilmelidir. Veri modelleme; veri yapıları, konum, zaman, gösterimi gibi çok boyutlu problem olarak görülmektedir.

Bu çalışmanın birinci bölümünde Christine Parent ve arkadaşları tarafından geliştirilmiş ve geçtiğimiz on yıl içinde uygulanmış MADS modelleme yapısı incelenmiş ve MADS uygulama alanları belirlenmiştir. İkinci bölümde, hem konuma hem de zamana bağlı platformların bir paket olarak kullanıcılara sunulmamasından dolayı, konuma bağlı yetenekleri olan bir veritabanındaki kavramlar verilmiştir. Bunun için Oracle 10g Spatial veritabanı kullanılmıştır. Veritabanında saklanan katmanlı Eminönü sayısal haritası bir grafik arayüzü ile görselleştirilmiştir. Üçüncü bölümde, hem konuma hem de zamana bağlı özellikleri olan bir hareket eden nesneler uygulamasının modelleme ve uygulama çalışmaları verilmiştir.

2 MADS Modeli

MADS modelinde modelleme yapıları; nesne tipleri, özellikleri, metotları, ilişki tipleri, çeşitli gösterimler, kısıtlar olarak sınıflandırılmıştır. MADS, çoklu algılama ve çoklu gösterimlere olanak sağlamaktadır[1]. Ayrıca, veri işleme ve sorgulama dili ile ilişkilendirilmiştir. Aşağıda bu kavramlar sırası ile açıklanmıştır.

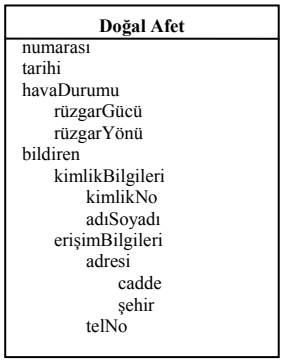
2.1 MADS Modelleme Yapıları

MADS modelinde kavramlar, nesneler veya o tipin örneği (instance) olarak isimlendirilir.Şekil 1'de numarası, kayıt tarihi ve sahibi özelliklerinden oluşan bir nesne gösterimi verilmiştir.

Şekil 1. Bir nesne şeması

Aynı özelliklere sahip sınıf örneklerini birbirinden ayırmak için sistem her nesneye bir nesne tanımlayıcısı(oid) atamaktadır. İlişkisel veritabanlarında nesne tanımlayıcısı birincil anahtar(primary key) olarak adlandırılmaktadır. Bir nesnenin basit ve karmaşık özellikleri olabilir. Örneğin yangın, çığ düşmesi gibi bir doğal afetin olduğu günün hava durumunu ve afeti bildirenen bilgilerini saklayan bir nesne yapısı Şekil 2'de verilmiştir.

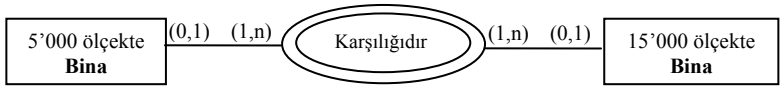
Buna göre numarası ve tarihi basit özellikleri; havaDurumu ve bildiren de karmaşık özellikleri olarak tanımlanmaktadır. Hava durumu, rüzgar gücü ve rüzgar yönü bilgilerinden oluşmaktadır.



Şekil 2. Basit ve karmaşık özelliklerden oluşan bir nesne şeması

Veritabanındaki nesneler modellendikten sonra, bunlar arasındaki ilişkiler belirlenmektedir. Her ilişkinin özellik sayısı(cardinality) en küçük ve en büyük sayı şeklinde iki sayı ile ifade edilir. Örneğin (0,n) sayıları; özelliğin seçimlik olduğunu, diğer bir ifade ile özelliğin hiç veya en fazla n tane olabileceğini, (1,n) sayıları da; özelliğin zorunlu olduğunu ve en az bir, en çok n tane olduğunu göstermektedir.

İlişkiler(relationship) gerçek dünyadaki bağlantıları göstermektedir. MADS ilişkiler, n'lidir($n \geq 2$) ve iki çeşittir; bağımlılık(association) ve çok-bağımlılık(multi-association). Bağımlılıklar da ikili olarak tanımlanmalıdır. Dairesel ilişkiler, mutlaka bir rol ile ilişkilendirilmelidir. Her ilişkinin bir numarası (rid) vardır. Nesne tipleri gibi ilişki tipleri de özelliklerle tanımlanmaktadır. Bazı durumlarda, haritanın ölçeğinden dolayı küçük ölçekte görülen 5 bina büyük ölçekte görülen 3 bina ile eşleştirilebilmektedir. Çok-bağımlılığa bir örnek Şekil 3'te verilmiştir.

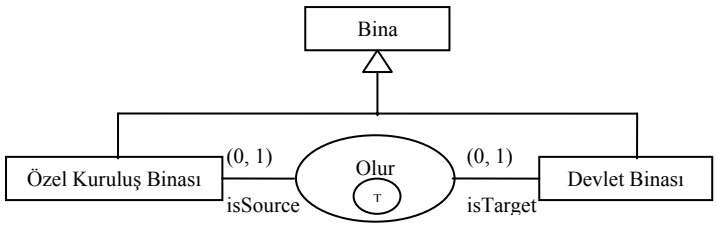


Şekil 3. Bir çok-bağımlılık ilişkisi

Şekil 3'teki çok bağımlılık, bir ilişkide gösterilen tek elips yerine iki iç içe elips ile gösterilmektedir. Şekil 3, haritada farklı ölçekte verilen bir bina diğer bir ölçekte gösterilmeyebilir veya eğer gösteriliyorsa mutlaka bir defa gösterilmelidir anlamına gelmektedir.

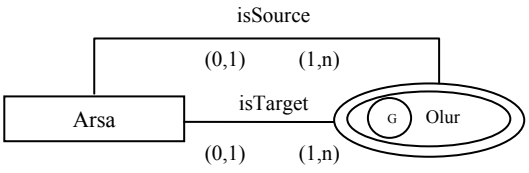
MADS modeli ile is-a ilişkisi modellenebilmektedir. Bazı uygulamalarda, standart kalıtımsallık ilişkisi alt sınıf tiplerde üst sınıf tiplerden gelen bilgilerin farklılaştırılması ile değiştirilebilmektedir. Ayrıca, tüm üst sınıfın özelliklerini alt sınıflarına bağlayan birleştirme(aggregation) ilişkileri de bu modelde gösterilebilmektedir. Tüm birleştirme ilişkileri ikili olmalıdır ve dairesel olabilir.

MADS modelinde sınıflar ve ilişkiler tamamlandıktan sonra, herhangi bir değişikliğin modele aktarılabilmesi için çeşitli anlamları olan iki yapı(semantics) geçiş ve yaratma yapıları da eklenmiştir[1]. Örneğin, bir ülkenin bir kasabasının zamanla gelişmesi sonucu, ülke yönetimi bu yeri şehir olarak tanımlayabilir. Bu, geçiş yapısı olarak tanımlanmıştır. Böylece zamanla değişen roller de model tarafından saklanabilmektedir.Şekil 4'te özel bir kuruluşun binasının, devlet binası olma durumuna geçme gösterilmiştir. (0,1) numaraları ile, devlet binasına dönüşen özel kuruluş binasının mutlaka önceden olması gerektiği belirtilmiştir.



Şekil 4. Geçiş yapısı

Yaratma yapısı ile varolan bir sınıf, zamanla iki sınıfa dönüşebilmektedir. Örneğin bir arsa, mirasçılar arasında paylaşıldıktan sonra daha küçük arsalara bölünmektedir. isSource ile yaratıcı rolü, isTarget ile yaratılan rolü gösterilmektedir. Şekil 5'te bu dairesel çok-bağımlılık ile gösterilmektedir.



Şekil 5. Yaratma yapısı

2.2 MADS Modelleme Veri Yapıları

Konum-zamana bağlı veritabanlarında nesneler, konuma ve zamana bağlı olarak belirtilebilmektedir. Bunu gerçekleştirebilmek için veritabanlarında konuma ve zamana ait veri tiplerinin tanımlanması gerekmektedir. Tablo 1'de konuma bağlı veri tipleri gösterilmiştir[1]. Burada pointbag, noktalardan oluşan bir diziyi göstermektedir.

Tablo 1. Konuma bağlı veri tipleri

Spatial data types	Icon	Spatial data types	Icon
Geo		SimpleGeo	
Point		Line	
OrientedLine		Surface	
SimpleSurface		ComplexGeo	
PointBag		LineBag	
OrientedLineBag		SurfaceBag	
SimpleSurfaceBag			

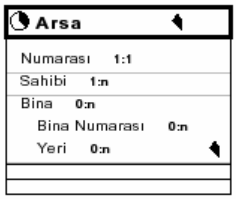
Zamana bağlı veri tipleri bir an veya bir aralık olabilir. ComplexTime, kompleks zamanı göstermekte ve hem bir andan hem de bir zaman aralığından oluşabilmektedir. Instantbag, anlardan oluşan bir dizidir. TimeSpan ise bir olayın başlangıç ve bitiş zamanlarını tutmaktadır. Zamana bağlı veri tipleri, Tablo 2'de verilmiştir.

Tablo 2. Zamana bağlı veri tipleri

Temporal data types	Icon	Temporal data types	Icon
Time		SimpleTime	
Instant		Interval	
ComplexTime		InstantBag	
IntervalBag		TimeSpan	

MADS ile konuma ve zamana bağlı olarak ilgilenilen bir olay, belirli veri tipleri kullanarak kesikli(discrete) veya sürekli(continuous) olarak tanımlanabilir. Eğer özellik sürekli ise bu, veri tipinin fonksiyonu olarak gösterilmektedir. Kesikli görünümde, uzay

alanları (regions) veya zaman aralıkları otonom varlıklar olarak yer alır. Şekil 6'da bir konum-zaman gösterimi verilmiştir. Binaya ait yer bilgileri konuma bağlı özellik olarak ifade edilmekte ve zamana göre arsa sahibi, numarası bilgileri değişmektedir.



Şekil 6. Hem konuma hem de zamana bağlı gösterim

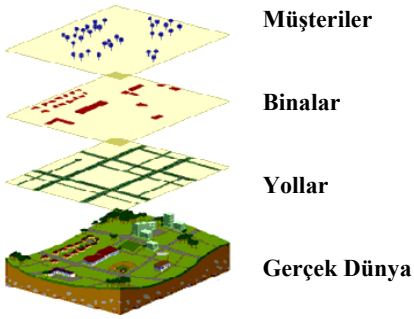
Nesneler veya ilişkiler için, zamana bağlı işlemlerde iki bilginin saklanması gerekmektedir. İşlem(transaction) ile bilginin ne zaman veritabanında saklandığı veya veritabanından silindiği ifade edilir. Buna karşın geçerli zaman ise uygulamaya göre bir olayın ne zaman geçerli olduğu verisi tutulmaktadır. MADS geçerli zaman bilgisini uygulamalara sağlamaktadır.

Bunlara ek olarak MADS modeline, nesnelerin konuma ait topolojik ve zamana bağlı ilişkilerin senkronizasyon gösterimleri de eklenerek model tamamlanmaktadır. Nesneler topolojik olarak birbirinden ayrı, çakışıyor, içinde, yaratıyor, değiyor, kesiyor ve eşit olarak uzayda görülebilir[4,5]. Zaman ilişkileri olarak da önce, esnasında, aynı anda başlıyor, eşit, biri biterken diğeri başlıyor, kesişiyor, aynı anda bitiyor ve ayrı tanımlanmıştır.

MADS modeli çoklu algılama ve gösterim için, farklı görünümlere bir stamp(pul) atamaktadır. Örneğin bir veri üç farklı şekilde algılanıyorsa, bu s1, s2 ve s3 ile gösterilmektedir. Son olarak, her özelliğin hangi pul tarafından algılandığı modele eklenmiştir.

3 Uygulama

Haritaların sayısallaştırılması için, birbiri ile ilişkili bilgiler gruplanarak bir katman olarak tanımlanır. Örneğin bir organizasyon, müşterilerini sayısal harita üzerinde görmek istiyorsa, Şekil 7'de verilen gerçek dünyadaki resme göre yolları, binaları ve müşterileri sırasıyla sayısal haritaya eklemektedir[2, 3].



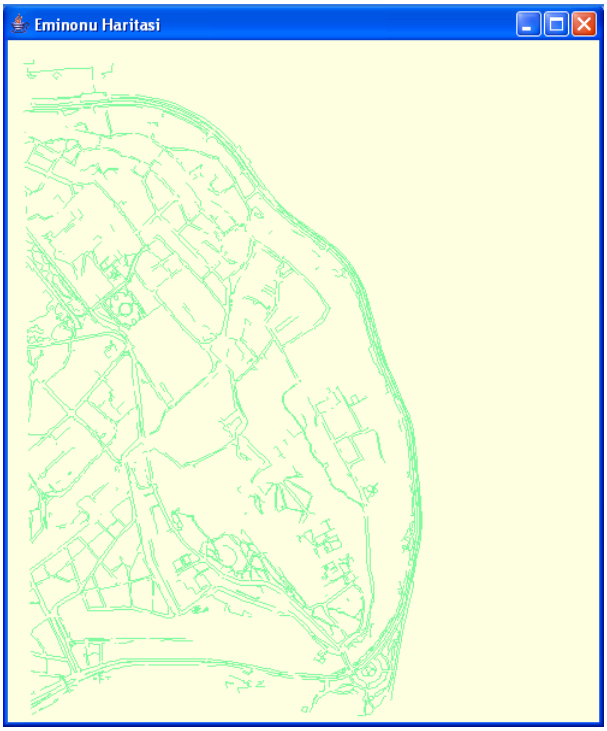
Şekil 7. Harita Katmanları

Yolları çizebilmek için çizgi, binaları çizebilmek için poligon ve müşterileri çizebilmek için de nokta geometrik yapılarından faydalanılabilir.

Konum bilgisine dayalı veritabanları, nesnelerin konuma ait özelliklerini saklayabilmek için çeşitli geometri özelliklerden yararlanır. Uygulama için seçilen Oracle 10g veritabanında point, line string, polygon, heterogeneous collection, multipoint, multilinestring ve multipolygon geometri tipleri SDO_GEOMETRY nesnesinde tanımlanmıştır[6]. Oracle veritabanı haritada bulunan her katmanı ayrı bir tabloda saklamaktadır. Dolayısıyla her katmanın geometri tipine göre veriler tablolardan okunabilir.

Sayısal harita bilgileri Oracle veritabanında, Oracle Spatial Java Library (SDOAPI) ile işlenmektedir. Bu kütüphane ile konum verileri okunabilmekte ve değiştirilebilmektedir. Verilerin sayısal olarak alınması, bir harita arayüzü gerçekleştirimek için yeterli olmamaktadır. Bu verilerin bir grafik arayüz ile görselleştirilmesi gerekmektedir. Gerçekleştirilen uygulamada Eminönü sayısal haritası, Oracle 10g'nin izin verdiği geometriler ve Java 3D kütüphanesi ileŞekil 8'deki gibi bir arayüzle görüntülenmiştir.

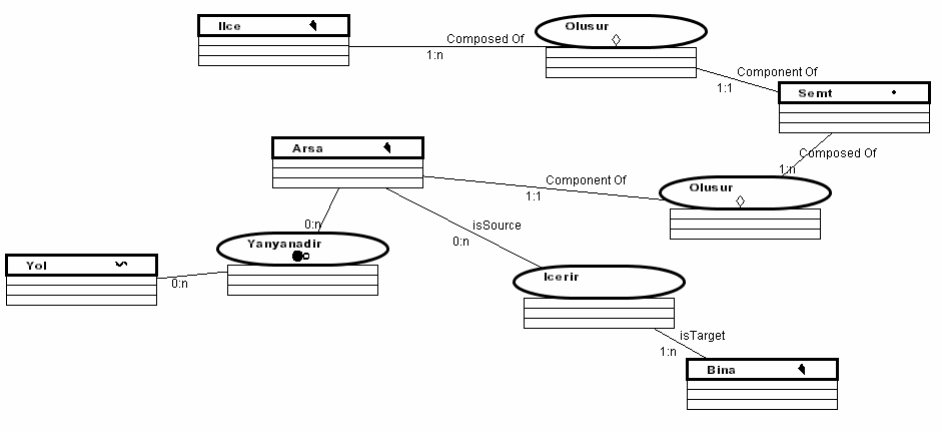
Şekil 8'deki sayısal haritanın üzerine hareket eden nesneler rastgele oluşturulmuştur. Hareket eden nesnelerin ve sayısal haritanın görüntülenmesi belirli zaman aralıklarında tekrar edilmektedir. Hareketli nesnelerin bilgileri de belirli zaman aralıkları ile veritabanına kaydedilmektedir.



Şekil 8. Eminönü sayısal haritası

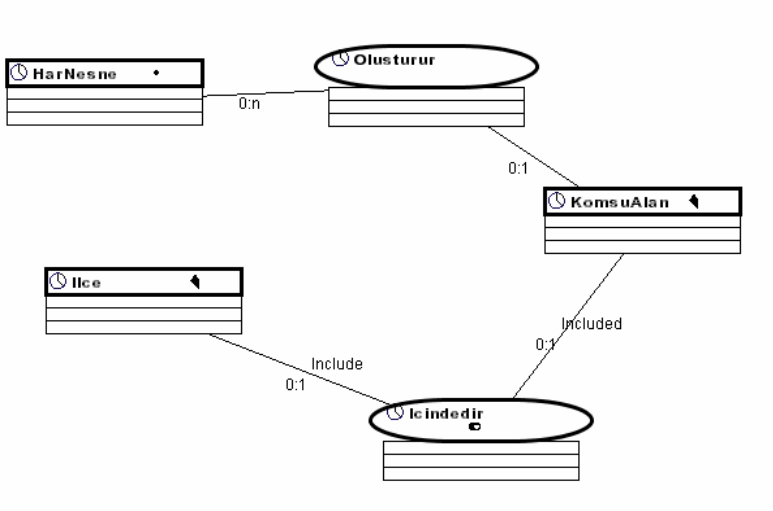
Gerçekleştirilen sistemde hareketli nesneler, kullanıcının girdiği yer bilgilerine göre sorgulanabilmektedir. Ayrıca veritabanında zamana bağlı ilişkiler olmadığı için, bu ilişkiler gerçekleştirilmiştir. Seçilen bir alan içinde, hangi nesnenin önce hareket etmeye başladığı, hangi nesnenin hareketinin sonlandığı gibi çeşitli sorgulamalar gerçekleştirilmiştir. Ayrıca seçilen nesneye en yakın komşu hareketli nesne de sistem tarafından belirlenebilmektedir. Bu sistemin MADS modeli Şekil 9'da verilmiştir.

Harita üzerinde poligon geometri tipinden oluşan ilçeler bulunmaktadır. Bir ilçede, birden fazla semt bulunabilir. Semtler nokta geometrisinden oluşmakta ve harita üzerinde nokta olarak semtler gösterilmektedir. Semtler, poligon geometrisinden oluşan arsalardan oluşmaktadır. Arsalar ile yollar harita üzerinde, yan yana bulunmaktadır. Yol, çizgi geometrisinden oluşmaktadır. Arsa üzerinde bir bina olabilir veya olmayabilir. Binalar poligon geometrisinde oluşmaktadır.



Şekil 9. Gerçekleştirilen sistemin MADS modeli

Şekil 10'da hareket eden nesneler için MADS modeli verilmiştir. Buna göre birden fazla hareketli nesne, en yakın komşuluk alanını oluşturmaktadır. Bu komşuluk alanı da ilçe haritasının içinde bulunmaktadır. Bu modül için belirli bir zaman veya aralık içinde hareket eden nesneler sorgulanabilmektedir.



Şekil 10. Hareket eden nesneler modülünün MADS modeli

4 Sonuç

Bu çalışmada hareket eden nesneler için konum ve zaman boyutunun da eklenmesi ile modelleme ihtiyaçları belirtilmiştir. Daha sonra uzun yıllardır konum ve zamana göre modelleme için kullanılan MADS modelleme yapısı ve veri tipleri incelenmiştir. MADS modeli kullanılarak, gerçekleştirilen uygulamada modellemenin daha kolay ve görsel olarak gerçekleştirildiği tespit edilmiştir. Konum bilgisine dayalı veritabanının kullanımı ile uygulamada yer bilgileri sorgulanabilmiştir. Zaman boyutu da eklenerek uygulama bir hareketli nesne uygulamasına dönüştürülmüştür.

Kaynakça

1. S. S. Christine Parent, Esteban Zimányi, Conceptual Modeling for Traditional and Spatio-Temporal Applications, Berlin, Heidelberg, Springer, 2006.
2. Perihan Kilimci ve Oya Kalıpsız, "Moving Objects Databases in Space Applications", in 3rd International Conference on Recent Advances in Space Technologies, 2007, İstanbul, s.106-108.
3. Perihan Kilimci ve Oya Kalıpsız, "Geometric Modelling and Visualization of Moving Objects on a Digital Map", in 2nd International Conference on Geometric Modelling and Imaging, 2007, Zurich, s.39-43.
4. Mokbel, M.F.,"Continuous Query Processing in Spatio-Temporal Databases", in EDBT 2004 Workshops, 2004, LNCS 3268.
5. Wolfson, P.S., B. Xu, J. Zhou, ve S. Chamberlain, "DOMINO: Databases fOr MovINg Objects tracking", SIGMOD '99, 1999: p. 547-549.
6. Oracle, Oracle Spatial User's Guide and Reference 10g Release 2 (10.2), January 2006, Oracle.

Ayrık Etkileşim Gruplarını Kullanan Yeni Bir Sınıf Uyum Belirleme Önerisi

Özcan Kurubaş[1], Nevcihan Duru[2]

[1,2] Kocaeli Üniversitesi, Bilgisayar Mühendisliği Bölümü, İzmit, Kocaeli
[1] ozcank@kou.edu.tr, [2] nduru@kou.edu.tr

Özet. Uyum (cohesion), nesneye yönelik sistemlerde, bir sınıfın üyelerinin ilişki derecesine işaret etmektedir. Yapılan çalışmalarda, uyumun tasarım kalitesi açısından önemli bir faktör olduğu kabul edilmektedir. Bundan dolayı, bir sınıfın uyumu tanımlanabilmeli ve ölçülebilmelidir. Literatürde, sınıf uyumunu, üyeler arasındaki bağlantılara göre yakalamayı amaçlayan çeşitli ölçütler önerilmiştir. Bunlar genellikle, metotların eriştikleri nesne değişkenleri ya da nesne değişkenleri paylaşan metot çiftlerini saymışlardır. Bu kriterler tek başına uyumun belirlenmesinde kısıtlayıcıdırlar. Mevcut ölçütler bazı özel metotları ve sınıf üyeleri arasında oluşabilecek ayrık etkileşim gruplarını dikkate alma hususunda yetersiz kalmaktadırlar. Bu çalışmada, sınıf üyeleri arasındaki etkileşimlere ve etkileşim yoğunluğuna odaklanan, üyeler arasında oluşabilecek ayrık grupları ve özel metotları dikkate alan metot tabanlı yeni bir kriter sunulacaktır. Ayrıca, sınıf uyumunun ölçülmesi hakkında yeni bir yaklaşım önerilecektir.

1 Giriş

Günümüzün yazılım uygulamaları daha karmaşıktırlar ve yazılım hataları daha kritik bir öneme sahiptirler. Çünkü yazılımda meydana gelebilecek bir hata ekonomik kayıplara ve hatta insan hayatını tehdit eden sonuçlara sebep olabilmektedir. Bundan dolayı, yazılımın kalitesini belirleyebilmek çok önemlidir. Daha kaliteli sistemlerin inşa edilmesi, son yıllarda tüm yazılım mühendisliği çabalarını yönlendiren amaç olarak karşımıza çıkmaktadır.

Nesneye yönelik geliştirme ortamları bu türden kaliteli sistemlerin yapımı için oldukça yol kat etmiştir. Fakat, bu ortamlar tek başına, kaliteli sistemler geliştirilmesinde her sorunu çözememektedirler. Örneğin, bir çok nesne ve nesneler arası bir çok ilişki içeren sistemlerde, karmaşıklık hızlı bir şekilde artış göstermektedir. Bu nedenle, geliştirme ortamına eklenmesi gereken ölçütlere gereksinim duyulmaktadır. Tüm bu ölçütler uygun sistemlerin geliştirilmesinde yardımcı olmaktadırlar. Böylece, yazılım ölçütleri yazılımın kalitesini belirlemenin vasıtası olarak önerilmektedirler [1].

Yazılım ölçütleri, yazılım geliştirme sürecini ve yazılımın kalitesini değerlendirmede nicel bir anlam sunmaktadırlar. Böylece yazılımın kalitesine değer biçmede önemli birer yardımcıdırlar. Örneğin, çalışmalar, yazılım ölçütleri ile yazılım kalite özellikleri olan hataya meyillilik ve bakım kolaylığı arasında ilişki olduğunu ortaya koymuştur [2], [3].

Bu çalışma aşağıdaki gibi yapılandırılmıştır: 2. bölümde uyum kavramı ve mevcut uyum ölçütleri hakkında genel bilgi verilmiştir. Bölüm 3 içinde, önerilen yaklaşıma dair

ayrıntılar sunulmuştur. Yeni yaklaşım ve gerekli tanımları bölüm 4 içinde verilmiştir. Önerilen yaklaşımın teorik değerlendirilmesi bölüm 5 içinde yapılmıştır. 6. bölümde sonuç sunulmuştur.

2 Uyum ve Sınıf Uyum Ölçütleri

Uyum, bir bileşen içindeki üyelerin birbirleri ile ilişki derecesine işaret etmektedir. Yüksek uyum, arzu edilen bir özelliktir. Yüksek uyumun daha fazla bakım kolaylığı ve daha fazla tekrar kullanım sağladığı geniş bir kabul görmektedir [3], [4], [5], [13].

Nesne tabanlı paradigma içinde, sınıf uyumu sınıf üyeleri arasındaki ilişkililiğin ölçüsü olarak düşünülmektedir[4]. İlişkililiğin sınıf açısından anlamı, sınıf tarafından sunulan metotların benzerliğidir. Bundan dolayı, eğer bir sınıfın üye metotları benzer fonksiyonellik sunuyorsa, sınıf uyumludur, benzer olmayan fonksiyonellik sunuyorsa, sınıf az uyumludur denir.

Bir sınıf, birbirleri ile ilişkisiz üyeler kümesi olmamalıdır, sınıfın tüm üyeleri ilgili nesnelerin bazı davranışlarını sağlamak için birlikte çalışmalıdırlar[16]. Uyumlu bir sınıfı parçalara ayırmak zor olmalıdır. Düşük uyuma sahip bir sınıf, birbirleri ile ilişkisiz üyelere sahiptir ve kolaylıkla parçalanabilir. Az uyuma sahip bir sınıf, uygun olmayan ya da rastlantısal bir soyutlamayı işaret etmektedir. Sınıf uyumu nesneye yönelik tasarımda önemli özelliklerden biri olarak düşünülmektedir. Uyum, tasarım süresince sürekli düşünülmesi gereken öncelikli amaçlardan biridir. Bir çok popüler çalışma, uyumu iyi tasarlanmış sınfların belirleyicisi olarak kullanmaktadır.

Literatürde, sınıf uyumunu belirlemek için çeşitli ölçütler önerilmiştir. Bugüne kadar sunulan ve kategorize edilen başlıca uyum ölçütleri [5]'de yer almaktadır. Ölçütler, nesne değişkeni kullanımı ya da nesne değişkeni paylaşımı tabanlıdırlar. Bu ölçütler, bir sınıf içindeki üyeler arasındaki bağlantılar anlamında sınıf uyumunu elde etmişlerdir. Bunlar, metotlar tarafından kullanılan nesne değişkeni sayılarını ya da nesne değişkenleri paylaşan metot sayılarını hesaplamışlardır. Bu türden ölçütlerin çoğu bugüne kadar denenmiş ve geniş bir şekilde literatürde tartışılmıştır [2], [4].

Bir çok araştırmacı sınıf uyumunu, önerdikleri kendi ölçütleri ile tanımlamışlardır. Önerilen ölçütlerin bir çoğu, Chidamber ve Kemerer tarafından tanımlanan LCOM (Lack of Cohesion) ölçütünden esinlenmiştir [10]. Bir çok araştırmacı, LCOM ölçütünü tekrar tanımlamışlardır. Bir sınıf, kendi nesne değişkenlerinden çoğu bir metotu tarafından kullanılıyorsa [5], [9] ya da bir çok sayıda metot çifti nesne değişkenleri paylaşıyorsa, çok uyumludur [3], [4], [8], [10], [12]. Tablo 1, ölçüt tanımlarının bir listesini vermektedir.

Chidamber ve Kemerer, sınıf uyumuna değer biçmek için LCOM (LCOM1 ve LCOM2) metriğini önermiştir. LCOM2, paylaştıkları ortak özellik olmayan metot çiftleri sayısından, en azından bir tane paylaştıkları ortak özellik olan metot çiftleri sayısının çıkarılması ile hesaplanmaktadır. LCOM2, eğer değeri negatif ise 0'a eşit kabul edilmektedir. LCOM1'in tekrar tanımlanması ile oluşturulan LCOM2, literatürde oldukça geniş olarak tartışılmıştır [5], [9], [12]. Li ve Henry, LCOM'u tekrar tanımlamışlardır [3]. LCOM3 olarak tanımladıkları ölçüt, metotların ayrık kümelerinin sayısıdır. Her bir küme sadece en azından bir özellik paylaşan metotları kapsamaktadır.

Hitz ve Montazeri LCOM3'ü tekrar tanımlamışlardır [11]. Bu ölçüt, grafik teorisi tabanlıdır ve bir grafikteki bağlı bileşenlerin sayısı olarak tanımlanmıştır. Grafikteki köşeler, sınıfın metotlarını temsil etmektedir. Eğer ilişkin metotlar aynı nesne değişkenine erişiyor ise, iki köşe arasında bir kenar yer almaktadır. Hendersen-Sellers LCOM5'i önermişlerdir [9]. LCOM5, kullanılan nesne değişkeni sayısı tabanlıdır. Bir sınıf, kendi nesne değişkenlerinin büyük bir kısmına kendi metotu tarafından başvuruda bulunuyorsa, daha uyumludur. Briand et al. LCOM5 için tekrar bir tanımlama sunmuştur [5].

Tablo 1. Kabul görmüş ölçütler

LCOM1	Ortak nesne değişkeni kullanmayan metot çiftlerinin sayısıdır [10].
LCOM2	P, nesne değişkenleri paylaşmayan metot çiftlerinin sayısı ve Q nesne değişkenleri paylaşan metot çiftlerinin sayısı ise bu durumda eğer |P|>|Q| ise LCOM2=|P|-|Q|, aksi halde LCOM2=0 olur [8].
LCOM3	Köşeleri metotlar, metotlar en azından bir nesne değişkeni paylaşıyor ise aralarında kenar olan yönlendirilmemiş bir G grafiği düşünüldüğünde, LCOM3=|G'nin bağlantılı bileşenlerinin sayısı| [3].
LCOM4	LCOM3'e benzer, G grafiğine ek olarak, M_i metotu, M_j metotunu işletiyorsa metotlar arasına kenarlar çizilir [11].
Co	LCOM4'teki G grafiğinin köşeleri V, kenarları E olsun. Bu durumda $Co = 2 . \frac{\| E \| - (\| V \| - 1)}{(\| V \| - 1) . (\| V \| - 2)}$ (Co=Connectivity) [11].
LCOM5	M_i (i=1,..,m) metot kümesinin A_j (j=1,..,a) nesne değişkeni kümesine eriştiğini düşünelim. $\mu(A_j)$, A_j nesne değişkenini kullanan metot sayısı olsun. $LCOM5 = \frac{(1/a)\sum 1 \le j \le a \ \mu(Aj) - m}{1 - m}$ [9].
Coh	LCOM5'in bir çeşididir. $Coh = \frac{\sum 1 \le j \le a \ \mu(Aj)}{m x a}$ [13].
TCC	Tight Class Cohesion. N adet metota sahip bir sınıf düşünüldüğünde, NP, azami metot çiftlerinin sayısını vermektedir: NP=[N*(N–1)]/2. NDC, metotlar arasında doğrudan bağlantıların sayısı olursa, TCC, doğrudan bağlı yerel metotların göreli sayısını verir:TCC = NDC / NP [4].
LCC	Loose Class Cohesion. NIC metotlar arasında doğrudan ya da dolaylı bağlantıların sayısı olursa, LCC, doğrudan ya da dolaylı bağlı metotların göreli sayısını verir: LCC=NIC/NP [4].

Bieman ve Kang, TCC ve LCC ölçütlerini sunmuşlardır [4]. Onlarda ortak değişken kullanan metot çiftlerini kullanmışlardır. Bir nesne değişkeni, bir metot tarafından doğrudan ya da dolaylı olarak kullanılabilir. Eğer bir nesne değişkeni, Mi metotunun gövdesinde bulunuyor ise, bu nesne değişkeni Mi metotu tarafından doğrudan kullanılmaktadır. Eğer bir nesne değişkeni, Mj tarafından doğrudan kullanılıyor ise ve Mj' de Mi tarafından doğrudan ya da dolaylı olarak işletiliyorsa, bu nesne değişkeni Mi metotu tarafından dolaylı olarak kullanılmaktadır. Eğer iki metot, doğrudan ya da dolaylı olarak ortak bir değişken kullanıyor iseler, bu iki metot, doğrudan ilişkilidir.

TCC, doğrudan ilişkili metot çiftlerinin göreli oranı olarak tanımlanmaktadır. LCC, doğrudan ya da dolaylı olarak ilişkili metot çiftlerinin göreli oranı olarak tanımlanmaktadır.

Son zamanlarda, araştırmacılar, değerleri sınıf içindeki diğer nesne değişkenleri üzerinden hesaplanan bağımlı nesne değişkenlerinin etkisini katarak mevcut ölçütleri geliştirmeyi hedeflemişlerdir [14], [15]. Bir kısmı ise sınıf içindeki uyumlu kısımların boyutları üzerinde çalışmışlardır [7].

3 Yaklaşıma Dair Ayrıntılar

Mevcut ölçütlerinden bazıları (LCOM2 gibi) uyumu, farklı bir değer beklendiği halde minimum olarak hesaplamaktadırlar. Bazıları (LCOM3 ve LCC gibi) ise üyeleri bir şekilde ilişkili sınıfa daima maksimum uyum değerini atamaktadırlar. Halbuki, sınıfların bazıları daha sık etkileşime, farklı ilişki desenlerine ve daha güçlü ilişki yoğunluklarına sahip olabilmektedirler. Üyeleri arasında daha fazla kullanım ilişkisi olan sınıflar doğal olarak daha yoğun ilişki desenine sahiptirler ve böylece fazla uyum değerine sahip olmalıdırlar. Sınıf üyeleri arasındaki etkileşim desenini ve ilişkinin gücünü dikkate almadan basit bir sayma yaklaşımı yukarıda bahsedildiği gibi çeşitli problemlere sebep olmaktadır. Fakat mevcut ölçütler, bunu dikkate almamaktadırlar. Bu çalışmada bu durumu dikkate alan bir yaklaşım sunulmaktadır.

Metotlar bazı nesne değişkenleriyle etkileşim içindedirler. Bazı metotlar sadece bazı nesne değişkenleri üzerinde, onlara erişimi organize etme gibi işlemler yaparlar. Bu metotlar önemli bir yere sahiptirler çünkü kapsülleme kavramı bu metotların kullanımını zorunlu tutar. Uyum ölçüm sürecine, bu metotların ilave edilmesi önemlidir. Aksi halde, yaklaşım, uyumluluğu doğru olarak değerlendiremeyecektir. Bu tür metotlar özel metotlar olarak adlandırılır. Erişim ve delegasyon metotları özel metotlar olarak sınıflandırılabilir. Literatürde, özel metotlara farklı şekillerde davranılmış genellikle ölçüm dışı bırakılmıştır [5], [6]. Bu durum uyumun beklenilen değerlerden farklı olmasına sebep olmuştur. Bu çalışmada, özel metotlar, erişim ve delegasyon metotlarına işaret etmektedir. Çalışmada, sınıf uyumu belirlenmeden önce özel metotlar için bazı kontrol işlemleri yapılmaktadır. Bu işlemler metotun özel metot olup olmadığının belirlenmesi ile ilgilidir.

Ayrıca, sınıf içinde, farklı fonksiyonellik sunan gruplar ortaya çıkabilmektedir. Başka bir deyişle, sınıfın bazı üyeleri sınfın diğer üyeleri ile etkileşimde bulunmaz. Bu durumda sınıf içinde ayrık etkileşim grupları oluşur. Uyum sınıfın tek bir fonksiyonellik sunmasına dayandığı için ölçütlerin bu durumu dikkate almaları gerekmektedir. Fakat mevcut ölçütler, sınıf farklı fonksiyonellik sunan gruplar içersede sınıf uyumunu yüksek olarak ölçmektedirler. Halbuki, ayrık gruplar uyumu kötü yönde etkilemelidir. Çünkü bazı metot ve nesne değişkenleri birbirlerinden bağımsız olarak sınıf içinde yer almaktadırlar ve bu uyumun doğasına aykırıdır. Bu çalışmada, ayrık grupların etkisi önerilen yaklaşım ile ölçüm sürecine katılmaktadır. Ayrık grup etkisi olarak belirlenen bir etki, tüm sınıf bazında ve metotların içinde bulundukları grup bazında metotları farklı olarak etkileyerek uyumunun oransal olarak azalmasına sebep olmaktadır. Eğer tek grup var ise metot uyumları değişmemekte, fakat grup oluşması halinde uyumlar azalma yönünde etkilenmektedir.

4 Uyum Ölçülmesine Dair Yeni Bir Yaklaşım

Tanım 1. Nesneye yönelik bir sistem, bir sınıf kümesinden oluşmaktadır, C. Her bir sınıf c Є C şeklinde tanımlıdır. c sınıfı bir küme nesne değişkeni A(c) ve bir küme metot M(c) içermektedir.

$$\mathrm{A(c)} = \{a_1, a_2, \ \ldots \ , a_j\}$$
$$\mathrm{M(c)} = \{m_1, m_2, \ \ldots \ , m_i\}$$

Metotlar, sınıf fonksiyonelliğinin kaynağıdır. Bu çalışmada, metotlar ayrıca sınıf uyumunun kaynağı olarak kabul edilmektedir. Uyumlu metotlar sınıfı daha uyumlu, uyumsuz metotlar sınıfı uyumsuz hale getirirler. Metot uyumu ise doğrudan ya da dolaylı olarak nesne değişkenlerinin kullanılması olarak tanımlanmaktadır.

Tanım 2. Bir mi metotu için, AD(mi), mi tarafından doğrudan erişilen nesne değişkeni kümesidir ve AI(mi), mi tarafından metot işletimleri vasıtası ile dolaylı olarak erişilen nesne değişkeni kümesidir.

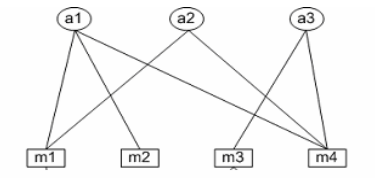
Metot tarafından doğrudan erişilen nesne değişkeni sayısı artarsa, metotun uyumu artar. Metot tarafından dolaylı olarak erişilen bir nesne değişkeni metot uyumuna katkı yapar, fakat bu katkı göreceli olarak daha azdır.

Tanım 3. Bir ai nesne değişkeni için, MD(ai), ai nesne değişkenine doğrudan erişen metot kümesidir.

Metot uyumu nesne değişkeni kullanım oranına bağlıdır. Metotlar tarafından çok kullanılan bir nesne değişkeni, onu kullanan metotları daha uyumlu yapar. Bir sınıfın uyumu, üyeler arasındaki etkileşimlerin sayısı yanında üyeleri arasındaki etkileşim desenine de bağlıdır. Başka bir deyişle, sınıf üyeleri arasındaki bağlılık uyumu etkiler. Nesne değişkenleri ve metotlar uyuma farklı ağırlıklarda katkıda bulunurlar.

Tanım 4. Bir c sınıfı için, metot-nesne değişkeni kullanım grafiği GMA(c), c'nin üyeleri arasındaki etkileşimin bir gösterimidir ve GMA(c)=(V,E) şeklinde yönlendirilmemiş bir grafik olarak tanımlanır.

$$\mathrm{V} = A(c) \cup M(c)$$
$$\mathrm{E} = \{(\mathrm{m,a}) \mid a \in A_D(m), m \in M(c), a \in A(c)\}$$



Şekil 1. Metot-nesne değişkeni kullanım grafiği

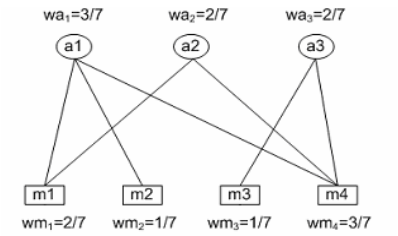
Örneğin, Şekil 1 metot-nesne değişkeni kullanım grafiğini göstermektedir. Grafik m1, m2, m3 ve m4 metotları ve a1, a2 ve a3 nesne değişkenleri arasındaki etkileşimleri yani etkileşim desenini göstermektedir.

Tanım 5. mi için metot ağırlığı, WM(mi), mi'nin doğrudan kullandığı nesne değişkeni sayısının, sınıftaki tüm metotların ayrı ayrı doğrudan kullandığı nesne değişkeni sayılarının toplamına oranı şeklinde tanımlanır. Bir metot hiçbir nesne değişkeni kullanmıyorsa, metotun ağırlık değeri 0 değerine eşit olur (Şekil 2).

$$WM(m_i) = \begin{cases} 0, if \quad A_D(m_i) = \phi \\ \dfrac{|A_D(m_i)|}{\sum_{m_j \in M(c)} |A_D(m_j)|} \end{cases} \tag{1}$$

ai için nesne değişkeni ağırlığı, WA(ai), ai'ye doğrudan erişen metot sayısının, sınıftaki tüm nesne değişkenlerine ayrı ayrı doğrudan erişen metot sayılarının toplamına oranı şeklinde tanımlanır. Eğer bir nesne değişkeni hiçbir metot tarafından kullanılmıyorsa, nesne değişkeninin ağırlık değeri 0 değerine eşit olur (Şekil 2).

$$WA(a_i) = \begin{cases} 0, if \quad M_D(a_i) = \phi \\ \dfrac{|M_D(a_i)|}{\sum_{a_j \in A(c)} |M_D(a_j)|} \end{cases} \tag{2}$$



Şekil 2. WM ve WA örnekleri

Şekil 2, bir metot-nesne değişkeni kullanım grafiğinindeki üyelerinin ağırlık dağılımlarını göstermektedir.

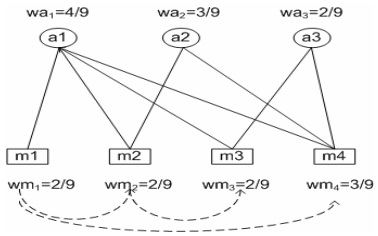
Tanım 6. mi için temel metot uyumu, BCOM(mi), mi'nin ağırlığı ile mi'nin doğrudan kullandığı nesne değişkenlerinin ağırlıklarının çarpımı şeklinde tanımlanır.

$$\mathrm{BCOM}(\mathrm{m_i}) = WM(m_i) \times \left(\sum_{a_j \in A_D(m_i)} WA(a_j) \right) \quad (3)$$

Sınıf üyeleri arasındaki etkileşim arttıkça etkileşim deseni de yoğunlaşır ve üyelerin uyuma katkı güçleri fazlalaşır. Basit bir sayma işleminden ziyade etkileşim deseni ve ilişki güçlerinin dikkate alınması uyumun daha etkin bir şekilde elde edilmesini sağlar. Bir metotun uyumu ayrıca nesne değişkenlerinin dolaylı olarak kullanımından da etkilenmektedir. Bir nesne değişkeninin bir metot tarafından dolaylı olarak kullanılıyor olması için, o nesne değişkenine o metotun başka metot işletimleri ile ulaşması gerekmektedir.

Tanım 7. mi ve aj için dolaylı erişim metot işletim yolu, MIP(mi, aj), mi'den aj'ye erişmek için mi tarafından doğrudan ya da dolaylı olarak işletilen metot kümesi şeklinde tanımlanır. Eğer birden fazla yol var ise, MIP(mi, aj) en kısa yolu içermelidir. Eğer tüm yollar eşit ise, MIP en yüksek dolaylı kullanım kriterine (IUC) sahip yolu içermelidir.

Tanım 8. aj için dolaylı kullanım kriteri, IUC(mi,aj), aj'nin mi uyumuna katkısının derecesidir ve MIP(mi,aj) yolundaki metotların BCOM değerlerinin çarpımı şeklinde tanımlanır.



Şekil 3. MIP örneği

Örneğin, Şekil 3'e göre, m1 a2'yi doğrudan kullanmamaktadır. Örneğe göre MIP içinde iki yol bulunmaktadır. Birinci yol (m1,m2) ve ikinci yol (m1,m4). Bunlar eşit uzunluktadır. Bu durumda, IUC değeri en büyük olan seçilir. Örneğe göre, uygun yol (m1,m4), çünkü bu yol için IUC değeri 6/81 ve diğeri için 4/81 olarak hesaplanmaktadır.

Bir metotun AD(m) kümesi tek elemanlı, AI(m) kümesi boş küme, AD(m) kümesinde yer alan nesne değişkeninin MD(m) kümesi boş küme ve m metotu sınıfta yer alan diğer metotlardan en az birinin MIP yolunda yer alıyorsa, o metot erişim ya da delegasyon metotu gibi özel bir metot olabilir. Bu durumda, özel metot ölçüm dışında tutulur ve ona yapılan çağrılar doğrudan nesne değişkenine yapılmış gibi davranılır. Sonuç olarak, bu aşamaya kadar yapılan hesaplama işlemleri tekrarlanır. Sonuç olarak, metot uyumu, ayrık gruplar içermeyen GMA için (4) kullanılarak bulunabilir.

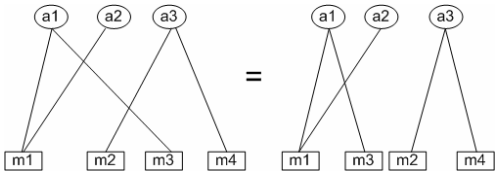
$$\mathrm{BCOM(m_i)} + \sum_{a_k \in A_I(m_i)} IUC(m_i, a_k) \tag{4}$$

GMA, üyeler arasındaki etkileşimlerin bir gösterimidir. Bazen, bir sınıf ayrık etkileşim desenine sahip olur, böylece GMA içinde farklı ayrık gruplar oluşur. Diğer bir deyişle, sınıfın bazı üyeleri sınfın diğer üyeleri ile etkileşimde bulunmaz. Bu sebeple, GMA alt metot-nesne değişkeni kullanım grafiklerine parçalanabilir. Doğal olarak, grafiğin içinde fazla grup olduğu durum istenmeyen bir durumdur ve uyumu azaltır.

Tanım 9. mi için ayrık grup etkisi, DGE(mi), ayrık grubun mi'ye olan etkisinin derecesidir. Ayrık grup içerisindeki mi metotunun BCOM değeri ile tüm dolaylı erişilen nesne değişkenleri için elde edilen IUC değerlerinin toplamı şeklinde tanımlanır. Eğer GMA'da sadece bir grup tanımlı ise, tüm metotların DGE değerleri 1 değerine eşit olur.

Her grup içindeki her metot için, DGE değeri hesaplanır. Bir metot için DGE değerinin hesaplanması, tek gruplu GMA içindeki metot için yapılan hesaplamadan farksızdır. Aradaki tek fark, hesaplama işlemine katılan metot sayıdır. Sonuç olarak, mi için DGE, (5) kullanılarak bulunabilir. Şekil 4, ayrık gruplara sahip bir GMA'yı göstermektedir.

$$\mathrm{DGE(m_i)} = \mathrm{BCOM(m_i)} + \sum_{a_p \in A_I(m_i)\ in\ DGE} IUC(m_i, a_p) \tag{5}$$



Şekil 4. G_{MA} içindeki farklı gruplar

Tanım 10. mi için metot uyumu, COM(mi), metotun son uyum değerini gösterir. mi'nin BCOM değeri ile tüm IUC değerleri toplamının DGE değeri ile çarpımı şeklinde tanımlanır.

$$\mathrm{COM(m_i)} = (\mathrm{BCOM(m_i)} + \sum IUC(m_i, a_k)) \times DGE(m_i) \tag{6}$$

Tanım 11. ci için sınıf uyumu, COC(ci), sınıfın üyeleri arasındaki ilişki derecesidir. Sınıfta bulunan tüm üye metotların uyum değerlerinin toplamı şeklinde tanımlanır.

$$\mathrm{COC(c_i)} = \begin{cases} 0, if \quad A_D(m) = \phi \wedge A_I(m) = \phi \quad \forall m \in M(c) \\ \sum_{m_k \in M(c_i)} COM(m_{k),else} \end{cases} \tag{7}$$

5 Yaklaşımın Teorik Olarak Değerlendirilmesi

Yaklaşım aşağıdaki gibi özelliklere sahiptir ve bu özellikleri ile genel olarak tüm sınıflar için uyumu belirlemede etkindir:

- Sınıflar için COC 0 ile 1 arasındadır ve ölçeklenebilir bir yapıya sahiptir.
- Her metot için, $M_D(m)$ kümesi ve $M_I(m)$ kümesi boş ise, o sınıf için COC 0 değerini alır. Her metot tarafından doğrudan kullanılan nesne değişkeni sayısı azami ise, o sınıf için COC 1 değerini alır.
- Sınıfa bir ilişki eklendiğinde COC değeri artar. Çünkü, etkileşim deseni yoğunlaşır. İlişki eklendikten sonra sırası ile (3), (4) ve (6) kullanılarak değerler hesaplanırsa değerin arttığı gözlenebilir.
- İlişkisiz sınıflar tek bir sınıf içinde toplandığında, COC değeri azalır. Bu durumda WM ve WA değerleri değişecektir. Bu değişiklik azalma yönünde olacaktır, bu (1) ve (2)'de görülebilmektedir.
- COC değeri, farklı sınıflar için farklı veya aynı değerleri verebilmektedir. Önerilen yaklaşım tüm sınıflar için kullanılabilir niteliktedir.
- Ölçüm sürecinde tanımsız aralıklar, kopuk süreçler bulunmamaktadır.
- Nesneye yönelik sistemlerin karakteristik özelliklerini dikkate almakta ve sınıf içinde oluşabilecek farklı ayrık etkileşim gruplarının uyumu azaltıcı etkisini doğal olarak sonuca yansıtabilmektedir.

6 Sonuç

Mevcut uyum ölçütleri özellikle nesne değişkeni kullanımı tabanlıdır. Bu kriter önemlidir, fakat bir sınıfın üyeleri arasındaki tüm ilişkileri yakalamak açısından yetersizdir. Sınıf içindeki bazı metotlar, özel metot olarak adlandırılan, doğal olarak sadece bazı nesne değişkenleriyle etkileşim içindedirler ve bundan dolayı uyumu azaltmamalıdırlar. Sınıf üyeleri arasındaki etkleşimlerin desenleri de önemlidir ve etkileşim desenini dikkate almadan sadece üyeler arasındaki ilişkilerin sayılması uyumun beklenilenden farklı elde edilmesine neden olmaktadır. Ayrıca, sınıfın etkileşim deseni içindeki ayrık gruplar sınıf uyumunu etkiler. Ayrık gruplar dikkate alınmadan elde edilen uyum değeri doğruyu yansıtmayacaktır ve karar verme aşamasında tasarımın başarısızlığına neden olacaktır. Bunlar neden mevcut uyum ölçüm yaklaşımlarının üyeler arasındaki ilişkiyi yansıtmada yetersiz kaldıklarını göstermektedir.

Bu çalışmada uyumun daha etkin bir şekilde ölçülmesi için, sınıfların karakteristiklerini yakalayan metot tabanlı yeni bir yaklaşım sunulmuştur. Bu yaklaşım, üyeler arasındaki etkileşim desenlerine odaklanmaktadır. Özel metotları tanımlayarak etkilerini aza indirecek bir süreç tanımlamaktadır. Ayrıca ayrık grupların etkisini uyum sürecine yansıtmaktadır. Bu şekilde uyumun elde edilmesi için farklı bir yaklaşım sunulmakta ve teorik olarak uygunluğunu gösterilmektedir. Bu çalışma yaklaşımın deneysel olarak doğrulanması ve çeşitli sistemlerde mevcut ölçütler ile birlikte kullanılarak etkinliğinin ispat edilmesi ile geliştirilebilir.

Kaynakça

1. Pressman, R.S., Software Engineering, A Practitioner's Approach, McGraw-Hill, 2001.
2. V. Basili, L. Briand, and W.L. Melo, “A Validation of Object-Oriented Design Metrics as Quality Indicators”, IEEE Transactions on Software Engineering 22, 1996, pp. 751-761.
3. W. Li and S. Henry, “Object-Oriented Metrics That Predict Maintainability”, Journal of Systems and Software 23, 1993, pp. 111-122.
4. J.M. Bieman and B.K. Kang, “Cohesion And Reuse In An Object-Oriented System”, Proceedings of The Symposium on Software Reusability (SSR’95), Seattle, April 1995, pp. 259-262.
5. L.C. Briand, J. Daly, and J. Wurs, “A Unified Framework For Cohesion Measurement In Object-Oriented Systems”, Empirical Software Engineering 3 (1), 1998, pp. 67-117.
6. K. El Emam, and W. Melo, “The Prediction Of Faulty Class Using Object-Oriented Design Metrics”, National Research Council of Canada NRC/ERB 1064, 1999.
7. H. Aman, K. Yamasaki, H. Yamada, and M-T. Noda, “A Proposal Of Class Cohesion Metrics Using Sizes Of Cohesive Parts”, Knowledge-Based Software Engineering, IOS Press, September 2002, pp. 102-107.
8. S.R. Chidamber, and C.F. Kemerer, “A Metrics suite for object Oriented Design”, IEEE Transactions on Software Engineering Vol. 20 No. 6, June 1994, pp. 476-493.
9. Henderson-Sellers, B., Object-Oriented Metrics Measures of Complexity, Prentice-Hall, 1996.
10. S.R. Chidamber, and C.F. Kemerer, “Towards a Metrics Suite for Object-Oriented Design, Object-Oriented Programming Systems”, Languages and Applications (OOPSLA), Special Issue of SIGPLAN Notices Vol. 26 No. 10, October 1991, pp. 197-211.
11. M. Hitz, and B. Montazeri, “Measuring Coupling And Cohesion In Object Oriented Systems”, Proceedings of the Int. Symposium on Applied Corporate Computing, October 1995, pp. 25-27.
12. L.C. Briand, S. Morasca, and V.R. Basili, “Defining and Validating High-Level Design Metrics”, Technical Report CS-TR-3301-1, University of Maryland, 1996.
13. H.S. Chae, Y.R. Kwon, and D.H. Bae, “A Cohesion Measure For Object-Oriented Classes”, Software Practice And Experience No. 30, 2000, pp. 1405-1431.
14. H.S. Chae, Y.R. Kwon, and D. H. Bae, “Improving Cohesion Metrics for Classes by Considering Dependent Instance Variables”, IEEE Transaction on Software Engineering 30(11), 826-832, 2004.
15. Y. Zhou, B. Xu, J. Zhao, and H. Yang, “ICBMC: an improved cohesion measure for classes”, Proceedings of International Conference on Software Maintenance, 44-53, 2002.
16. N.E. Fenton and S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Company, 1998.

Yazılım Ürün Hattı Mimarisi Tasarım ve Analiz Yöntemleri Üzerine Bir İnceleme

A. Egemen Yılmaz[1]

[1] HAVELSAN A.Ş. Barış Kartalı Özel Proje Müdürlüğü, Mustafa Kemal Mah., Ankara.
[1] asimegemenyilmaz@yahoo.com

Özet. Yazılım ürün hattı, ürünlerde yeniden kullanımın en üst seviyede tutulması, hatta yeniden kullanımın sadece ürünlerle sınırlı kalmayıp geliştirme süreçlerine de yansıması amacıyla ortaya atılmış bir kavramdır. Yazılım ürün hatlarında, normal ürünlerden farklı bir takım kalite nitelikleri bulunmaktadır; dolayısıyla yazılım ürün hattı mimari tasarım ve analiz yöntemleri de, tek bir ürüne yönelik olarak geliştirilmiş olan yöntemler ile farklılıklar göstermektedir. Bu çalışmada, özellikle yazılım ürün hatları için geliştirilmiş veya yazılım ürün hatlarına uyarlanabilir olan mimari tasarım ve analiz yöntemlerinden, yazarın bilgisi dâhilinde olanlar incelenmiştir.

1 Giriş

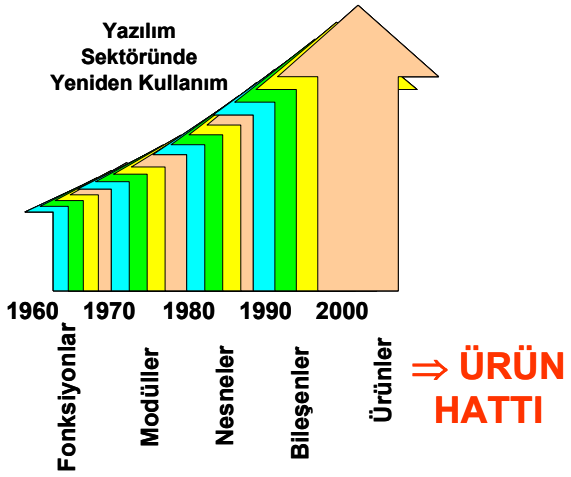
Yazılım ürün hattı, yazılım üreticisi kuruluşların hızlı bir şekilde yüksek kalite ve düşük maliyet ile aynı anda birçok ürün geliştirebilme ve idame edebilmeleri amacıyla ortaya atılmış ve özellikle son yıllarda hayli yaygınlaşmış bir kavramdır. Yazılım ürün hattı, "belirli bir pazara/alana yönelik olarak, ortak temel bileşenler kullanılarak geliştirilen/geliştirilmiş ve birçok ortak özelliği/gereksinimi bulunan yazılım ağırlıklı ürün kümesi" olarak tanımlanabilir [1].

Yazılım sektöründe "yeniden kullanım", özellikle ekonomik nedenlerden dolayı 1960lı yıllardan bu yana giderek daha sistemli ve kapsamlı bir şekilde gerçekleştirilen bir faaliyet halini almıştır. Şekil 1'de de görüldüğü üzere, 40-50 yıl önce fonksiyon bazında gerçekleştirilmekte olan yeniden kullanım faaliyeti; günümüzde "ürün"lerin ve hatta bu ürünlerin geliştirilmesi esnasında kullanılan süreçlerin yeniden kullanımı boyutlarına ulaşmış olup; ürün hattı kavramının doğmasına yol açmıştır.

Bu ana değin yapılmış olan saha çalışmalarında, yazılım ürün hatlarının hızlı bir şekilde (×10'a varan boyutlarda) ve yüksek kalitede (/10'a varan hata oranları ile) ürün çıkarma; düşük maliyetli geliştirme ve bakım-idame (%60'a varan geliştirme ve bakım-idame maliyeti indirimi); kaynakların etkin kullanımı (/10'a varan personel sayısı ile ürün çıkarabilme); etkin paydaş (ihtiyaç makamı, onay makamı, müşteri, ana yüklenici, alt yüklenici vb.) yönetimi; hızlı ve kolay ürün portföyü geliştirme; ürünler/sistemler arası uyumlu çalışabilme gibi faydalar sağlamakta olduğu gözlenmiştir [2].

Yazılım ürün hattı yaklaşımlarının sadece belli sektörlerde, belirli ölçekteki şirketlerde veya belirli ürün tiplerinde başarılı ve faydalı olabileceğine; ayrıca ürün hattı yaklaşımlarının yazılım kalite süreçleri ile çelişmekte olduğuna dair yanlış kanılar günümüzde de devam etmektedir. Ancak ürün hattı yaklaşımı, farklı boyutlarda ve

değişik uzmanlık alanına sahip birçok yazılım şirketinde; Komuta Kontrol Sistemleri'nden Telekomünikasyon Santralleri'ne, Web Tabanlı Piyasa Analiz Yazılımları'ndan Kurumsal Yönetim Sistemleri'ne birçok farklı ürün ailesinde, bilinen yazılım kalite süreçleri altında başarı ile uygulanmıştır. Öte yandan, faydaları ancak orta ve uzun vadede belirginleşen bu yaklaşımın herhangi bir kurumda başarı ile uygulanabilmesi için, kurum içerisinde üst yönetimden teknik ekiplere kadar yaygınlaşmış bir kültür oluşmuş olması gerektiği de gözlenmiştir.



Şekil 1. Yazılım Sektöründe Yeniden Kullanım Yaklaşımları – Tarihsel Gelişim.

2 Yazılım Ürün Hattı Mimarileri

Yazılım mimarisi, en temel anlamda "bir yazılımın organizasyonel kırılımı; bu kırılımdaki yazılım bileşenleri; bu bileşenlerin dışarıdan gözlemlenebilir özellikleri ve bu bileşenlerin birbirleri ile ilişkileri"nden oluşan bir bütündür [3]. Ürün hattı kavramının tanımı ve doğası gereği, yazılım mimarisinin önemi çok büyüktür. Şekil 2'de, yazılım mimarisinin ürün hattı kavramı içerisindeki yeri görülmektedir [2].

2.1 Yazılım Kalite Nitelikleri

Gerek bir yazılımın mimari tasarımı, gerekse bu mimari tasarımın analizi esnasında; türevlenebilirlik, esneklik, yeniden kullanılabilirlik, güvenilirlik, idame edilebilirlik gibi birçok "kalite niteliği"nin göz önünde bulundurulması ve mimari tasarımın çok yönlü bir şekilde incelenmesi gerekmektedir. Tasarımı veya analizi yapılacak mimari, bir yazılım ürün hattına ait olduğunda ise; söz konusu kalite nitelikleri kümesinde "ürün hattı" hattı kavramına özgü bir takım eklentiler ve değişiklikler yapmak gerekmektedir.

Ayrıca, bazı kalite nitelikleri ortak/temel veya tüm bileşenler bazında, bazıları ise tüm mimari/altyapı bazında ele alınmalıdır. Ürün hattına özgü kalite nitelikleri ve bunların tanımları, aşağıdaki şekilde özetlenebilir [4]:

- Değişkenlik (Bileşen bazında): Bir bileşenin, ürün hattındaki değişik ürünler için ortak özelliklerinin bulunması,
- Türevlenebilirlik (Ortak / temel bileşen bazında): Bir bileşenin, ürün hattındaki bir ürüne mahsus özelliklerinin bulunması,
- Yeniden Kullanılabilirlik (Ortak / temel bileşen bazında): Bir bileşenin, ürün hattındaki değişik ürünlerde kullanılabiliyor olması,
- Değer Biçilebilirlik (Bileşen bazında): Bir bileşenin "katma değerinin" hesaplanabiliyor olması,
- Entegre Edilebilirlik (Altyapı / mimari bazında): Bir ürüne mahsus bir özelliğin/bileşenin ürün hattı altyapısına entegre edilebilirliği,
- Doğruluk ve Bütünlük (Bileşen bazında): Bir bileşenin kendi gereksinimlerine ve ürün hattı temel görev tanımına uygunluğu,
- Evrimlendirilebilirlik (Bileşen bazında): Bir bileşenin sürekli değişim ve gelişime açık olması,
- Yönetilebilirlik (Hem bileşen, hem mimari / altyapı bazında): Değişik ihtiyaçlara yönelik değişimler durumunda "planlama, gözlemleme ve karar verme" faaliyetlerinin kolay yapılabilirliği,
- İdame Ettirilebilirlik (Bileşen bazında): Bileşen üzerinde düzeltici, uyumlandırıcı, önleyici veya iyileştirici bir bakım-idame faaliyetinin yapılabilirliği.

Şekil 2. Yazılım Mimarisinin Ürün Hattı Kavramı İçerisindeki Yeri.

2.2 Yazılım Ürün Hattı Mimari Tasarım Yöntemleri

Yazılım mimarisi tasarımı ve bu tasarımların gösterim yöntemleri, üzerinde sayısız konferans bildirisi, dergi makalesi ve birçok kitap yazılmış olan hususlardır. Günümüze

kadar birçok araştırmacı, farklı bakış açıları ile değişik birçok mimari tasarım yöntemi geliştirmiş olup; bunlardan yalnızca belirli birkaçı "ürün hattı" odaklı olmuştur. Yazarın bilgisi dâhilindeki yazılım ürün hattı mimari tasarım yöntemleri aşağıdaki gibidir:

- Bileşene Yönelik Platform Mimari Oluşturma Yöntemi (Component-Oriented Platform Architecting Method / COPA) [5],
- Aileye Yönelik Soyutlama, Spesifikasyon ve Çevirim Yöntemi (Family-Oriented Abstraction, Specification and Translation / FAST) [6],
- Özelliğe Yönelik Yeniden Kullanım Yöntemi (Feature-Oriented Reuse Method / FORM) [7],
- Bileşen Tabanlı Uygulama Geliştirme Yöntemi (Komponentbasierte Anwendungsentwicklung / KobrA) [8],
- Kalite Odaklı Mimari Tasarım ve Analiz Yöntemi (Quality Driven Architecture Design and Analysis / QADA) [9],
- Nitelik Odaklı Tasarım Yöntemi (Attribute Driven Design Method / ADD) [10],
- Ürün Hattı Yazılım Mühendisliği – Alana Özgü Yazılım Mimarisi Yöntemi (Product Line Software Engineering – Domain Specific Software Architecture (PuLSE-DSSA)) [11],
- Kalite ve Niteliğe Yönelik Yazılım Mimarisi Yöntemi (Quality Attribute Oriented Software Architecture / QASAR) [12].

Ürün hattı mimarisi tasarımında mimari merkezli ve bileşen tabanlı olmak üzere iki ana yaklaşım bulunmaktadır. Mimari merkezli yaklaşım, ürün/bileşen çeşitliliğine dayalı yukarıdan-aşağıya bir geliştirme yaklaşımı iken; bileşen tabanlı yaklaşım ise bileşenlerin dizilimine ve ilişkilerine dayalı aşağıdan-yukarıya bir geliştirme yaklaşımıdır. Yukarıda listelenmiş olan yöntemlerden bazıları mimari merkezli, bazıları bileşen tabanlı olup; belli başlı birkaç tanesi her ikisinin özelliklerini paylaşmaktadır.

Philips tarafından geliştirilmiş olan COPA, yoğunluklu olarak tercih edilen BAPO (İş, Mimari, Süreç ve Organizasyon (Business, Architecture, Process, and Organization)) [13] ürün hattı geliştirme yaklaşımına dayalı bir yöntemdir. Uygulama alanları daha çok telekomünikasyon ve biyomedikal sektörlerinde kullanımı olan yöntem, mimari merkezli ve bileşen tabanlı yaklaşımlar arasında bir denge kurmayı hedeflemektedir. Müşteri ihtiyaçlarının analizi aşamasında uygulanmaya başlayan COPA yöntemi, mimari tasarım aşamasında proje koşulları, tüm projelerin paydaş beklentileri, mevcut mimari(ler), mimari tasarım uzmanlarının tecrübeleri gibi girdileri kullanmaktadır. COPA'nın ana hedef ve amaçları, boyut, çeşitlilik ve karmaşıklık yönetimi; kalite artırımı; geliştirme sürelerinin düşürülmesi olarak özetlenebilir.

Ürün hattı konusunda literatüre büyük katkılar yapmış olan D. M. Weiss tarafından geliştirilmiş ve ilk olarak Lucent Technologies tarafından uygulanmış olan FAST yönteminin hedef kitlesi, ortak yönü (fonksiyonalite, arayüz, kod vb.) çok olan birçok ürünü aynı anda geliştirmeyi ve idame etmeyi hedefleyen kuruluşlardır. Yöntemin ana amaçları, benzer/ortak aktiviteleri indirgeyerek süreçleri etkinleştirmek, üretim maliyetlerini ve geliştirme sürelerini düşürmek olarak özetlenebilir. Yöntem, ürün hattı geliştirme süreçleri tamamen göz önünde bulundurularak tanımlanmış olup kolay

uyarlanabilirliği sayesinde ürün hattına özgü kalite niteliklerinin tasarım kriteri olarak dâhil edilmesine olanak sağlamaktadır.

Kore'de bulunan Pohang Bilim ve Teknoloji Üniversitesi'nde geliştirilmiş olan FORM yöntemi, Özelliğe Yönelik Alan Analizi (Feature-Oriented Domain Analysis (FODA)) [14] tekniğinin ürün hatları için genişletilmiş ve genelleştirilmiş halidir. Bilgi teknolojileri ve telekomünikasyon sektörleri hedeflenerek geliştirilmiş; ancak diğer sektörlerde ve hatta gerçek-zamanlı sistemlerde de başarıyla uygulanmış olan FORM'da temel hedef, ürün hattındaki "ortaklık" ve "değişkenlik" analizlerinden yola çıkarak yeniden kullanılabilir ve uyarlanabilir bileşenler geliştirmektir. Ürün hattına özgü kalite nitelikleri, yöntemin alan analizi fazında uygun bir şekilde tanımlanması durumunda ele alınabilmektedir.

Fraunhofer Deneysel Yazılım Enstitüsü tarafından geliştirilmiş olan KobrA yöntemi, bileşen-tabanlı ve artırımlı bir ürün (tek bir ürün veya ürün hattı) geliştirme yaklaşımıdır. Model Odaklı Mimari (Model Driven Architecture (MDA)) [15] uyumlu olan yöntemin, ürün hattı geliştirme süreçlerinin tamamını kapsamakta olmasına rağmen tasarımcı/geliştirici bakış açıları ile geliştirilmiş olduğu söylenebilir. Aslen bilgi sistemleri için önerilmiş olup, kolay uyarlanabilir yapısı sayesinde diğer sektörlerdeki projelerde de başarıyla uygulanmıştır.

Finlandiya Teknik Araştırma Merkezi VTT'de geliştirilmiş olan QADA, kalite odaklı bir yöntem olması nedeniyle, daha çok servis odaklı sistemler ve ara katman yazılımlarının tasarımları ve de analizleri için uygundur. Kalite odaklı olması dolayısıyla yöntem, kalite gereksinimlerinin ele alınması ve kalite niteliklerinin gözden geçirilmesi ile işe başlamaktadır. Yöntemin uygulanması ile elde edilen temel sonuçlar; kavramsal ve somut tasarımlar; ayrıca kalite gereksinimlerinin bu tasarımlar ile ne derecede sağlanmakta olduğunu gösteren analiz sonuçlarıdır. Yöntem, geliştirme ekibi içerisinde gereksinim mühendisinden test mühendisine, bakım idame sorumlusundan proje yöneticisine birçok farklı bakış açısı ile geliştirilmiş olmasına rağmen; kurumun organizasyon yapısı ve iş stratejileri gibi hususları ele almamaktadır.

Carnegie Mellon Üniversitesi Yazılım Mühendisliği Enstitüsü (CMU-SEI) ile R. Bosch GmbH tarafından geliştirilmiş olan ADD, kalite gereksinimlerinden yola çıkarak ve de kalite nitelikleriyle ilgili yönlendirmeler ile tasarımcıya yol göstermeyi ve de karar aldırtmayı hedefleyen bir yöntemdir. Yöntem her ne kadar kalite niteliklerini ele almayı amaçlıyorsa da, ürün hattına özgü kalite niteliklerinin tasarımcı tarafından ayrıca ve dikkatli bir şekilde göz önünde bulundurulması gerekmektedir.

Fraunhofer Deneysel Yazılım Enstitüsü tarafından yazılım mimari tasarımı ve analizi amacıyla geliştirilmiş olan PuLSE-DSSA, referans bir mimariyi temel alan, senaryo tabanlı ve tekrarlamalı bir yöntemdir. En genel ve en önemliden, en detaylı ve önemsize doğru sıralanmış senaryolar ile mimarinin değerlendirilmesi prensibinden yola çıkan yöntemin çıktıları, mimari tasarımın yanı sıra "mimari karar(lar) modeli"dir. Ürün hattına özgü kalite niteliklerinin, senaryolar içerisinde mutlaka ayrıca ele alınmalarını gerektiren yöntemde bileşen tasarımı, "ortaklık" ve "değişkenlik" analizleri sonucunda yapılmaktadır.

Ürün hattı konusunda literatüre büyük katkılar yapmış başka bir araştırmacı olan J. Bosch tarafından geliştirilmiş olan QASAR, kalite niteliklerinin ele alınması ve

tasarımın kalite gereksinimlerine uyumluluğunun değerlendirilmesine dayalı, kalite odaklı bir başka yöntemdir. Yöntemin ana aşamaları fonksiyonel ve kalite gereksinimlerinden yola çıkılarak bir mimarinin oluşturulması; senaryo bazlı simülasyonlar veya matematiksel modellemeler ile kalite niteliklerinin gözden geçirilmesi; elde edilen sonuçlara göre mimaride gerekli değişikliklerin yapılarak aynı işlemlerin tekrarlanması olarak özetlenebilir. Ürün hattına özgü kalite niteliklerinin yöntem tarafından ele alınabilmesi için, kalite gereksinimleri içerisinde mutlaka kapsanması gerekmektedir.

Literatürde, yukarıda listelenmiş olan tasarım yöntemleri içerisinden birer alt kümeyi karşılaştırmalı olarak inceleyen bir takım çalışmalar bulunmaktadır ([16], [17], [18], [19]). Bunların arasında Verdin ve Olalde'nin çalışması [18], FAST, FORM, KobrA, QADA, ADD, PuLSE-DSSA ve QASAR yöntemlerini; Matinlassi'nin çalışması ise [19], COPA, FAST, FORM, KobrA ve QADA yöntemlerini karşılaştırmalı olarak incelemektedir.

2.3 Yazılım Ürün Hattı Mimarisi Analizi Yöntemleri

Mimari tasarımların analizi, ilgili tasarımların farklı bakış açıları ile ele alınması ve değerlendirilmesini gerektirmektedir. Mimari tasarım yöntemlerinde olduğu gibi, analiz yöntemlerinde de "ürün hattı" odaklı yaklaşımların sayısı sınırlıdır. Yazılım ürün hatlarına uygulanabilir olan veya özellikle yazılım ürün hatları düşünülerek geliştirilmiş olan mimari analiz yöntemlerinden, yazarın bilgisi dâhilindekiler aşağıda verilmiştir:

- Senaryo Tabanlı Mimari Analizi Yöntemi (Scenario-Based Architecture Analysis Method / SAAM) [20],
- Karmaşık Senaryolar Üzerine Kurulu Senaryo Tabanlı Mimari Analizi Yöntemi (Scenario-Based Architecture Analysis Method Founded on Complex Scenarios / SAAMCS) [21],
- Alan-Spesifik Genişletilmiş Senaryo Tabanlı Mimari Analizi Yöntemi (Extending Scenario-Based Architecture Analysis Method in the Domain/ ESAAMI) [22],
- Evrimlendirilebilirlik ve Yeniden Kullanılabilirlik Odaklı Senaryo Tabanlı Mimari Analizi Yöntemi (Scenario-Based Architecture Analysis Method for Evolution and Reusability / SAAMER) [23],
- Mimari Al-Sat Analizi Yöntemi (Architecture Trade-Off Analysis Method / ATAM) [24],
- Senaryo Tabanlı Mimari Yeniden Yapılandırma Yöntemi (Scenario-Based Architecture Re-Engineering / SBAR) [25],
- Mimari Seviyede Bakım-İdame Efor Kestirim Yöntemi (Architecture Level Prediction of Software Maintenance / ALPSM) [26],
- Yazılım Mimarisi Ölçme/Değerlendirme Modeli (Software Architecture Evaluation Model / SAEM) [27],
- Mimari Seviyede Değiştirilebilirlik Analizi (Architecture Level Modifiability Analysis / ALMA) [28],
- Yazılım Mimarisi Performans Değerlendirmesi (Performance Assessment of Software Architecture / PASA) [29].

Esas olarak değiştirilebilirlik kalite niteliği odaklı geliştirilmiş olmasına rağmen; verilen bir yazılım mimarisinin tüm kalite niteliklerine uygunluğunu değerlendirmeyi hedefleyen SAAM'ın ideal kullanım aşaması, üst seviye tasarım ile gerçekleştirme fazları arasındaki dönemdir. Aday mimari tasarımların, olası mimari değişiklik durumlarını içeren senaryolar ile değerlendirilmesi prensibine dayanan yöntem, ilgili değişiklik işlemlerinin maliyetleri ve karmaşıklıkları açısından bir takım metrikler sunmakta ve potansiyel problem alanlarını belirlemektedir. Yöntem, 1993 yılından bu yana derleyici/hata giderici yazılımlardan bilgi sistemlerine, hava trafik kontrol sistemlerinden konfigürasyon yönetim yazılımlarına kadar birçok mimari tasarımın analizinde başarı ile uygulanmıştır. Sonraki yıllarda SAAM'ın değişik bakış açıları ile genişletilmesi sonucu SAAMCS, ESAAMI ve SAAMER gibi bir takım yeni yöntemler geliştirilmiştir.

SAAMCS, olası değişiklik senaryolarının karmaşıklığı üzerine de bir takım değerlendirmeler yapma ve metrikler tanımlama esasına dayanmaktadır. Senaryolarda, "değişiklik isteğini başlatan taraf" bilgisini de ayrıca ele alan yöntem, kalite nitelikleri arasından sadece esnekliği değerlendirmekte olup şu ana kadar bilgi sistemleri projelerinin mimari tasarım analizinde başarıyla uygulanmıştır.

ESAAMI, SAAM'ın alan uzmanlık bilgisinin yeniden kullanım bakış açısı ile genişletilmesi prensibine dayanmaktadır. Yöntem, yeniden kullanım durumlarının ele alınması amacıyla ortaya atılmış olan "protosenaryo" kavramı ve alan bilgisine dayalı bir takım ağırlıklandırmalar ile aday mimariler arasında bir karşılaştırma yapılması olarak özetlenebilir. Olgunluk seviyesi henüz diğer yöntemler kadar olmayan ESAAMI'yi geliştiren araştırmacılar, yöntemin "analiz esnasında yeniden kullanım bakış açısı nedeniyle analistin nesnelliği kaybetmesi" gibi bir risk taşıdığını, uygulama esnasında bu hususa dikkat edilmesi gerektiğini belirtmişledir.

Uygulanması esnasında alan uzmanlarının tecrübelerine ve konu ile ilgili girdilerine ihtiyaç duyan SAAMER ise, SAAM'ın evrimlendirilebilirlik ve de yeniden kullanım açısından genişletilmiş hali olarak düşünülebilir. Yöntemin en dikkat çekici özelliği, birbiri ile ilişkisi düşük olan değişiklik senaryolarında aynı bileşenlerin değişmesi gerektiği sonucunun çıkıp çıkmadığını değerlendirmesi; mimari tasarım içerisinde (varsa) böyle bileşenlerin sınırlarının doğru çizilip çizilmediğini sorgulamasıdır. Yöntem, şu ana değin telekomünikasyon projelerinde başarıyla uygulanmıştır.

İlk başta Carnegie Mellon Üniversitesi Yazılım Mühendisliği Enstitüsü'nün spiral tasarım modeli çalışmalarından türevlendirilerek sadece tasarım esnasında kullanılacak bir al-sat analiz yöntemi olarak önerilmiş olan ATAM, sonradan genişletilerek bir mimari tasarım ve analiz yöntemi haline getirilmiştir. Tasarım esnasında genellikle belirli kalite nitelikleri arasında bir al-sat analizi yapılması ve karar verilmesi gerektiği prensibinden yola çıkan yöntem, (geliştiricileri tarafından yazılım geliştirme sürecinin herhangi bir aşamasına uygulanabilir olduğu iddia edilmesine rağmen) ideal olarak mimari tasarımın sonlandırılmasını takiben uygulanmaktadır. Teorik olması itibarı ile diğer yöntemler gibi ağırlık, protosenaryo vb. bir takım özel tanımlar gerektirmeyen yöntem, öte yandan belirli aşamalarda paydaşların aktif katılımını ve gözden geçirme faaliyetlerini gerektirmesi, yoğun dokümantasyon ve geliştirme aracı kullanımı gibi ihtiyaçları dolayısıyla ağır bir süreç tanımlamaktadır. Aktif paydaş katılımı gerektirmesi nedeniyle, başarıyla uygulanmış olduğu askeri, web-tabanlı, gömülü platformlarda

geliştirilmiş değişik birçok projede sadece teknik anlamda değil; sosyal anlamda da fayda sağladığı gözlemlenmiştir.

Önerilen bir yazılım mimarisinin verilen yazılım kalite niteliklerine uygunluğunu senaryolar, benzetimler, matematiksel modellemeler ve tecrübeye dayalı gerekçelendirmeler ile ölçmeye dayalı bir yöntem olan SBAR, mimari tasarımı idame edilebilirlik, güvenilirlik, performans ve yeniden kullanım gibi birçok farklı bakış açısı ile değerlendirebilmektedir. SBAR, şu ana değin bir ölçme-kalibrasyon sisteminde başarı ile uygulanmıştır.

ALPSM ise, mimari tasarım değerlendirmesini herhangi bir bakım-idame faaliyetinin neden olacağı değişiklik ihtiyacı ve ilgili değiştirme eforu hesabı vasıtasıyla yapan bir yöntemdir. Uyarlayıcı ve mükemmelleştirici bakım-idame faaliyetlerine dayalı senaryoların önem ve gerçekleşme ihtimaline göre ağırlıklandırılması ve ilgili sonuçların karşılaştırılması, yöntemin esasını oluşturmaktadır. Şu ana kadar bir biyomedikal uygulamasında başarı ile uygulandığı bilinmektedir.

Kalite niteliklerine dayalı olarak tanımlanmış bir takım kalite metriklerinin hesaplanması ile nihai ürün kalitesinin değerlendirilmesi prensibine dayalı olan SAEM; hem kullanıcı, hem de geliştirici bakış açıları ile analiz yapmayı hedefleyen bir yöntemdir. Olgunluk seviyesi görece düşük olan yöntemin, yazarın bilgisi dâhilinde başarı ile uygulandığı bir sistem bulunmamaktadır.

ALMA yöntemi, zaman zaman ALPSM ile karıştırılıyor olmasına rağmen, aslen ALPSM ve SAAMCS yöntemlerini geliştirmiş olan araştırmacıların bir araya gelip her iki yöntemin güçlü yönlerini ele alarak geliştirmiş oldukları yeni bir yöntemdir. Bakım idame faaliyetlerinden kaynaklanacak durumlara ilişkin senaryolar oluşturmak; bu senaryolara dair değiştirilebilirlik ve risk analizleri yapmak, yöntemin esasını oluşturmaktadır.

Analiz edilecek olan aday mimari tasarımların değişik gösterimlerde dokümantasyonlarını gerektiren PASA, performans gereksinimleri başta olmak üzere idame edilebilirlik gibi belli başlı bir takım kalite nitelikleri açısından değerlendirmeler ve gerekli durumlarda al-sat analizleri yapabilmektedir. Yeni geliştirme projelerinin yanı sıra modernizasyon projelerinde de uygulanabilir olan yöntem, gerçek zamanlı sistemlerden web-tabanlı finansal yazılımlara kadar geniş bir yelpazede başarılı olmuştur.

Literatürde, mimari tasarım yöntemlerinde olduğu gibi, analiz yöntemlerinin de karşılaştırmalı olarak incelendiği bir takım çalışmalar bulunmaktadır ([30], [31], [32]). Dobrica ve Niemelä'nın çalışmaları SAAM, SAAMCS, ESAAMI, SAAMER, ATAM, SBAR, ALPSM ve SAEM yöntemlerini karşılaştırmalı olarak incelerken [30], [31]; Babar ve Gorton'un çalışması ise SAAM, ATAM, ALMA ve PASA yöntemleri ile sınırlı kalmıştır [32].

3 Sonuçlar ve Tartışma

Bu çalışma, yazılım ürün hattı mimari tasarım ve analiz yöntemleri hakkında bir derleme çalışması olarak düşünülebilir. Yazarın yazılım ürün hatları hakkındaki

tecrübeleri doğrultusunda ve bilgisi dâhilindeki yöntemler listelenmiş ve en basit anlamda bir karşılaştırma yapılmıştır.

Ürün hattı mimari tasarımı konusunda teknik ve mali anlamlarda etkin yöntemlerin geliştirilmesi, yazılım kurumlarının büyük önem vermekte olduğu çalışmalardır. Bu nedenle, ürün hattı mimari tasarım yöntemlerinin (Bölüm 2'de de görüldüğü üzere) genelde endüstriyel kuruluşlar tarafından ve uygulamalı olarak geliştirildiği gözlemlenmektedir. Analiz yöntemlerinin geliştirilmesi ise, şu ana değin daha ziyade akademik camianın ilgi alanı dâhilindeki kuramsal çalışmalar olarak kalmış; bazı yöntemler henüz bir uygulama alanı bularak olgunlaşma olanağına kavuşamamıştır.

Mimari tasarım yöntemleri arasında COPA, ürün hattı geliştirme süreçlerinin tamamını kapsıyor olması açısından öne çıkmaktadır. Özellikle ürün hattı odaklı olması ve değişik problemler için uyarlanabilirliği yüksek olmasına rağmen FAST, çok uygulanabilir bir yöntem olarak değerlendirilmemektedir [19]. FORM özelliğe yönelik; KobrA bileşen tabanlı; QADA, ADD ve QASAR ise kalite odaklı klasik mimari tasarım yaklaşımlarının ürün hattı kavramı için (mümkün olduğunca bire bir) uyarlanmış halleri olarak düşünülebilir. Hem tasarım, hem de analiz yöntemi olarak değerlendirilebilecek PuLSE-DSSA'in etkin olabilmesi için ise, PuLSE ailesine ait diğer prosedürlerle (PuLSE-ECO vb.) birlikte kullanılması gerekmektedir. Bütün yöntemler, ürün hattına özgü kalite niteliklerini bir aşamada ve bir şekilde girdi olarak kabul etmekte ve değerlendirmekte olup, bu açıdan ön plana çıkan bir yöntem bulunmamaktadır.

Analiz yöntemleri arasında ise SAAM ve ATAM, en olgun yöntemler olarak öne çıkmaktadır. ATAM, birden fazla kalite niteliğine odaklanmış olması; diğer yöntemlere göre daha fazla veri (al-sat analiz noktaları vb.) sunması; paydaş katılımını en üst seviyede tutması ile en kapsamlı yöntem olarak gözükmektedir [32]. Ancak, daha önce de belirtildiği gibi ağır bir süreç tanımlamakta ve gerektirmektedir. Fonksiyonel olmayan gereksinimlere ve özellikle yazılım mimari yapısına dengeli bir şekilde odaklanması açısından SBAR, umut vaat eden ve ilerleyen zamanlarda uygulama alanı buldukça daha çok atıf alacak olan bir yöntem olarak göze çarpmaktadır [31].

Ancak daha sağlıklı kanılara varabilmek için, hem tasarım, hem de analiz yöntemlerinin aynı problemler üzerinde, eşit şartlarda, nesnel bir şekilde, kapsamlı olarak değerlendirilmesi ve karşılaştırılması gerekmektedir. Bu amaçla bir takım kuramsal/uygulamalı çerçeveler geliştirilmesi ve ilgili yöntemlerin değerlendirilmesi, literatürde bu konudaki büyük bir boşluğu dolduracaktır.

Kaynakça

1. Clements, P., Northrop, L. M., Software Product Lines: Practices and Patterns, Addison-Wesley, 2002, ISBN: 0-201-70332-7.
2. Northrop, Linda M., "Software Product Lines: Reuse That Makes Business Sense", CMU-SEI Product Line Practice Presentations, http://www.sei.cmu.edu/productlines/SPLKeynote.pdf.
3. Bass, L., Clements, P. ve Kazman, R., Software Architecture in Practice, Addison-Wesley, 1998, ISBN: 0-321-15495-9.

4. Anastasopoulos, M., Bayer, J., “Product Family Specific Quality Attributes (IESE Report No. 042.02/E, Version 1.0)”, Fraunhofer Institut Experimentelles Software Engineering, 2002.

5. America, P. ve diğerleri, “COPA: A Component-Oriented Platform Architecting Method …”, Proceedings of the 1st Conf. on SW Product Line Engineering, 2000.

6. Weiss, D., Lai, C. ve Tau, R., Software product-line engineering: a family-based software development process, Addison-Wesley, 1999, ISBN: 0-201-69438-7.

7. Kang, K. C. ve diğerleri, “FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures”, Annals of Software Engineering, cilt 5, 1998, s. 143 - 168.

8. Atkinson C. ve diğerleri, Component-based product line engineering with UML, Addison-Wesley, 2002, ISBN: 0-201-73791-4.

9. Matinlassi, M., Niemelä, E., ve Dobrica, L., “Quality-driven architecture design and quality analysis method, …”, VTT (Technical Research Centre of Finland) Publications, 2002.

10. Bass L., Bachmann, F., “Architecture Based Design Method – Introduction”, CMU-SEI Product Line Practice Presentations, http://www.sei.cmu.edu/productlines/add_method.html.

11. Anastasopoulos, M., Bayer, J., Flege, O. ve Gacek, C., “A Process For Product Line Architecture and Evaluation: PuLSE-DSSA – Version 2.0 (IESE Report No. 038.00/E, Version 1.0)”, Fraunhofer Institut Experimentelles Software Engineering, 2000.

12. Bosch, J., Design and Use of Software Architectures - Adopting and evolving a Productline approach, AddisonWesley, 2000, ISBN: 0-201-67494-7.

13. van Ommering, R., “Building Product Populations with Software Components”, Proceedings of ICSE'02, 2002, s. 255 - 265.

14. Kang, K. C., Cohen, S., Hess, J., Novak, W., ve Peterson, A., “Feature-Oriented Domain Analysis - Feasibility Study (CMU/SEI-90-TR-21)”, CMU-SEI, 1990.

15. Frankel, D., Model Driven Architecture, Applying MDA to Enterprise Computing, Wiley Publishing Inc., 2003, ISBN: 0-471-31920-1.

16. Lopez-Herrejon, R., Batory, D., “A Standard Problem for Evaluating Product-Line Methodologies”, Proceedings of Generative and Component-based SW Engineering: 3rd Int. Conf. (GCSE 2001), (cilt. 2186, Lecture Notes in Computer Science, Springer Verlag), 2001.

17. M. Harsu, “A Survey of Product-Line Architectures”, Tampere University of Technology, Teknik Rapor 23, Mart 2001.

18. Verdin, K. C., Olalde, C. L., “Assessment of Product Line Architecture and Aspect Oriented Software Architecture Methods”, Proceedings of 1st First Workshop on Aspect Oriented Product Line Engineering (AOPLE), Ekim 2006.

19. Matinlassi, M., “Comparison of Software Product Line Architecture Design Methods: …”, Proceedings of the 26th Int. Conf. on SW Engineering (ICSE’04), 2004.

20. Kazman, R., Abowd, G., Bass, L. ve Clements, P., “Scenario-Based Analysis of Software Architecture”, IEEE Software, Kasım 1996, s. 47 - 55.

21. Lassing, N., Rijsenbrij, D., ve van Vliet, H., “On Software Architecture Analysis of Flexibility, …”, Proceedings of 2nd Nordic SW Architecture Workshop (NOSA’99), 1999.

22. Molter, G., “Integrating SAAM in Domain-Centric and Reuse-Based Development Processes”, Proceedings of 2nd Nordic SW Architecture Workshop (NOSA’99), 1999.

23. Lung, C., Bot, S., Kalaichelvan, K. ve Kazman, R., “An Approach to Software Architecture Analysis for Evolution and Reusability”, Proceedings of CASCON’97, Kasım 1997.

24. Kazman, R., Klein, M., Barbacci, M., Lipson, H., Longstaff, T. ve Carrière, S. J., “The Architecture Tradeoff Analysis Method”, Proceedings of ICECCS, Ağustos 1998, s. 68 - 78.

25. Bengtsson, P. O., Bosch, J., “Scenario Based Architecture Reengineering”, Proceedings of 5th Int. Conf. on SW Reuse (ICSR5), 1998, s. 308 - 317.

26. Bengtsson, P. O., Bosch, J., “Architecture Level Prediction of Software Maintenance”, Proceedings of 3rd Eur. Conf. on SW Maintenance and Reengineering, Mart 1999, s. 139 - 147.

27. Duenas, J. C., de Oliviera, W. L., de la Puente, J. A., “A Software Architecture Evaluation Model”, Proceedings of 2nd Int. ESPRIT ARES Workshop, Şubat 1998, s. 148 - 157.

28. Bengtsson, P.O., Lassing, N., Bosch, J. ve van Vliet, H., “Architecture-level modifiability analysis (ALMA)”, The Journal of Systems and Software, cilt 69, 2004, s. 129 - 147.

29. Williams, L. G., Smith, C.U., "PASA: A Method for the Performance Assessment of Software Architecture," Proceedings of the 3rd Workshop on SW Performance, 2002.

30. Dobrica, L., Niemelä, E., Strategy for Analysing Product Line Software Architectures, Espoo: VTT (Technical Research Centre of Finland) Publications, 2000.

31. Dobrica, L., Niemelä, E., “A Survey on Software Architecture Analysis Methods”, IEEE Transactions on Software Engineering, cilt 28, sayı 7, Temmuz 2002, s. 638 - 653.

32. Babar, M. A., Gorton, I., “Comparison of Scenario-Based Software Architecture Evaluation Methods”, Proceedings of the 11th Asia-Pacific SW Engineering Conf. (APSEC'04), Kasım-Aralık 2004, s. 600 - 607.

DURUM ÇALIŞMALARI

Çok Katmanlı Kurumsal Projelerde Uygulama Yaşam Döngüsü Yönetimi

Ceyhun Özgün[1], Sedat Akel[2]

[1,2] Cybersoft Enformasyon Teknolojileri Ltd. Şti,
ODTU Teknokent Silikon Binaları 1. Kat No: 18 06531 ODTU / Ankara
[1] ceyhun.ozgun@cs.com.tr, [2] sedat.akel@cs.com.tr

Özet. Kurumsal projelerde, büyüklüklerinden kaynaklanan iletişim ve yönetim sorunları, yazılım geliştirme ve uygulama yönetimi aşamalarında problemleri kaçınılmaz hale getiriyor. Dolayısıyla geliştirme ekipleri arasındaki koordinasyon, isteklerin ve değişikliklikliklerin izlenmesi, konfigürasyon ve sürüm yönetimi, geliştirilen yazılımların test süreçleri gibi çeşitli süreçlerin etkin olarak yerine getirilmesi proje başarısı açısından hayati öneme sahip oluyor. Ayrıca uygulamaların çalıştıkları ortamlara yayılması, çalışma zamanında versiyonlarının ve durumlarının kolaylıkla izlenip yönetilebilmesi de son zamanlarda geliştirme süreçleri kadar önem kazanmıştır.
Bu bildiride, Gelir İdaresi Başkanlığı (GİB) Vergi Dairesi Otomasyon Projesi (VEDOP) gibi, J2EE tabanlı çok katmanlı mimariye sahip kurumsal projelerde karşılaşılan yukarıda bahsedilen problemler gösterilecek ve son zamanlarda yaygınlaşan Uygulama Yaşam Döngüsü Yönetimi kavramları göz önünde bulundurularak geliştirilmiş olan CyberSoft Application Management Platform (CAMP) çözümü sunulacaktır.

1 Giriş

Yazılım projeleri genelde iletişim ve yönetim problemleri yüzünden başarısızlıkla sonuçlanmaktadır. Daha büyük olan kurumsal projelerde farklı alt projeler arası bağımlılıklar ve farklı proje ekipleri arasındaki iletişim zorlukları yüzünden bu problemler daha da yönetilmez hale gelmektedir.

Kurumsal projeler genellikle her birine proje denebilecek büyüklükte çeşitli alt projelerden oluşmaktadır. Bu projelerin birbirleri ile sıkı ilişkide olmaları kurumsal projelerin yönetimini daha da zorlaştırmaktadır. Kurumsal projeler artık sadece yazılım geliştirmeden ibaret ve sadece yazılım geliştiricilerin sorumlu olduğu bir iş olarak görülerek başarılı olamazlar. Projeler artık gereksinimlerin analizi aşamasından başlayarak, tasarım, geliştirme, test, kurulum ve yayım, izleme ve bakım aşamalarına kadar çeşitli grupların tek bir hedefe odaklanarak ortaya koydukları ortak bir çaba olarak görülmelidir. Bu yüzden ekipler arası iletişimin olabildiğince etkin hale getirilmesi gerekmektedir.

Ayrıca artık Şelale Modeli [1]'ndeki gibi bu aşamalardan tek bir seferde geçerek işler tamamlanamamaktadır. Tekrarlı süreçlerin [2] ve çevik yöntemlerin [3] ortaya çıkmasıyla her bir süreç projeye değer katacak şekilde izlenebilir ve yönetilebilir bir etkinliğe dönüştürülmüştür.

Uygulama Yaşam Döngüsü Yönetimi [4] gereksinimlerin toplanmasından bakım ve kurulu ortamda yönetim aşamalarına kadar olan tüm aşamaların tekrarlı olarak yönetilmesini ve aşamalar arasındaki iletişimin kolaylaştırılmasını sağlamaktadır.

2 Uygulama Yaşam Döngüsü Yönetimi (Application Lifecycle Management)

Uygulama Yaşam Döngüsü Yönetimi yaşam döngüsü aşamalarını, geri dönülmez birbirlerinden tamamen bağımsız aşamalar olarak görmek yerine işbirliği içinde çalışan tümleşik tekrarlı süreçler olarak görmektedir. Bu sayede ekipler arasındaki iletişim de kolaylaşmaktadır. Özellikle son dönemde yaygınlaşan farklı yerleşkelere dağılmış ekiplerin kullanıldığı dağıtık geliştirmenin kullanıldığı projelerde ortak çalışabilirlik oldukça artmaktadır.

Çeşitili Uygulama Yaşam Döngüsü Yönetimi aşamalarında yapılan işlemlerin rahatlıkla yerine getirilmesini sağlayan araçların çıkışı ile elle yapılan işlemlerin daha etkin olarak yerine getirilebilmesi sağlanmıştır. Tekrarlı olmaları ve elle yapılmaları nedeniyle hataya açık olan bu işlemler bu araçlar sayesinde hızlı ve hatasız olarak yapılabilmekte ve kolaylıkla yönetilebilmektedir.

İlk Uygulama Yaşam Döngüsü Yönetimi araçları her aşama için ayrı olarak tasarlanmış, farklı ihtiyaçlara cevap veren, farklı üreticilerin ürettiği ve aralarında ortak çalışma imkanı olmayan araçlardır. Her aşamada farklı aracın kullanılması ve aşamalar ve dolayısıyla araçlar arasında bilgi taşınması gerekliliği işleri zorlaştırmaktadır.

Son zamanlarda ALM 2.0 [15] ismi ile ortaya çıkan akım ile araç üreticileri her bir aşama için ayrı araç üretmek yerine tüm aşamaları birlikte yönetecek bütünleşik olarak çalışan araç ve araç aileleri geliştirmeye başladılar. Bu sayede proje ekipleri kendi işleri için kullandıkları aracı terketmeden diğer ekiplerle birlikte çalışabilme ve kolay iletişim imkanına kavuştular.

3 Uygulama Yaşam Döngüsü Yönetimi Teknikleri

3.1 Problem Yönetimi (Issue Management)

Küçük projelerde yazılım testleri sonrasında çıkan hatalar, bunların düzeltme istekleri, değişiklik ve yeni özellik istekleri kolaylıkla takip edilebilir. Kurumsal projelerde ise çok sayıda modül olması, geliştirme, test, oluşturma işlemlerinin farklı ekipler tarafından yapılması nedeniyle isteklerin takip edilmesi zorlaşmaktadır. Bu istekler, çeşitli Problem Yönetimi araçları ile kolaylıkla takip edilebilir. İsteğin ortaya çıkışından kapanışına kadar geçirdiği tüm aşamalar yönetilebilir ve izlenebilir. Kapalı isteklerin açık isteklere oranı, her bir sürümde çıkan hataların sayısı gibi çeşitli raporlar ile sürecin kalitesi ve proje ilerlemesi hakkında güvenilir bilgiye hızlı bir şekilde ulaşılabilir.

3.2 Sürüm Yönetimi (Release Management)

Birden fazla alt projenin aynı fiziksel ortamdaki sunucularda çalışması ve bu alt projelerin farklı gereksinimlere sahip olması ortak kullanılan modüllerin sürümlerini

takip etmeyi oldukça zorlaştırmaktadır. Sürüm Yönetimi, proje modüllerinin sürümlerinin ve modüllerde yapılan değişikliklerin hangi isteklerden kaynaklandığının, modüllerin hangi sürümlerinin hangi ortamlara yayıldığının takip edilebilmesini sağlar. Bu işlevi sağlayan araçlar yazılım sürüm yönetimi araçları ile bütünleşik olarak çalışmaktadırlar.

3.3 Otomatik Oluşturma (Automated Build)

Küçük projelerde proje ürünlerinin geliştiricinin kullandığı geliştirme ortamı içinde oluşturulması çoğunlukla yeterli olmaktadır. Ancak kurumsal projelerde çalışan sayısının çok olması, modül sahiplerinin değişmesi, modüle yeni geliştirici atanması gibi sebeplerden sıklıkla modüllerin yeni geliştiricilerin bilgisayarlarına kurulması gerekmektedir. Proje kaynak kodu dizini, kullanılan kütüphanelerin dizin ve versiyonları gibi ayarlar genellikle her bilgisayarda farklı olmakta bu da işlerin uzamasına sebep olmaktadır. Ayrıca test/çalışma ortamı ile farklı ayarların yapılması test ve çalışma ortamında problemler çıkarmaktadır. Proje ürünlerinin oluşturulması için bir oluşturma ortamı ayırıp, çeşitli oluşturma araçları kullanarak bu problemler çözülebilir. Otomatik Oluşturma araçları kaynak kodu ve kullandığı diğer modül ve üçüncü parti kütüphaneleri yazılım sürüm yönetimi araçlarından istenilen sürümleri ile alıp, kolaylıkla oluşturabilmektedir. Java [18] platformu için Apache Ant [5] gibi açık kaynak kodlu oluşturma araçları sayesinde oluşturma işlemleri platform bağımsız ve genişletilebilir bir hale getirilebilmektedir. Örneğin, oluşturma işlemi kaynak kodun analizi, birim testlerin çalıştırılması gibi çeşitli kotrolleri içerecek şekilde düzenlenebilmektedir.

3.4 Statik Kod İncelemesi (Static Code Analysis)

Yazılım hatalarının düzeltilmesinin, hatanın oluştuğu yaşam döngüsü aşamasından sonra atladığı her aşamada daha fazla maliyete sebep olduğu bilinmektedir. Özellikle kurumsal projelerde ortak kullanılan alt yapı modüllerinde ortaya çıkan hatalar tüm uygulamaların hizmet kalitesini düşürebilmektedir. Statik Kod İncelemesi [6], geliştiriciler tarafından sıklıkla yapılan boş göstergeç ataması (null pointer assignment), dizi index hatası gibi bulunması göreceli olan kolay hataların kod daha oluşturma aşamasında iken yakalanabilmesini sağlar. Hatalar kaynak kod veya derlenmiş kodun çalıştırılmadan incelenmesi ile bulunur. Bu inceleme işlemi otomatik oluşturma işlemine ek bir iş olarak tanımlanabilmekte ve her oluşturma işleminde kodun gözle incelenmesine gerek kalmadan sık karşılaşılan temel hatalardan arındırılması sağlanabilmektedir. Java platformu için açık kaynak kodlu Checkstyle [7], PMD [8], FindBugs [9] gibi araçlar bu amaçla kullanılabilmektedir.

3.5 Sürekli Birleştirme (Continuous Integration)

Birbirleriyle ilişkili alt projelerin birleştirme işlemleri genellikle sıkıntılı olmaktadır. Özellikle bu işlemler alt projelerin teslimine yakın zamanlara bırakıldığında çok büyük sorunlar yaşanmaktadır. Geliştirilen kodun sadece proje sonunda değil, ilk kodun geliştirilmesinden itibaren sıklıkla diğer kodlarla birleştirilmesi, çıkacak problemlerin daha önceden farkedilmesini ve daha masraflı olmadan çözülebilmesini sağlar. Sürekli Birleştirme [10] araçları, otomatik birleştirmeden bir adım ileri giderek kodların belirli

aralıklarla düzenli olarak otomatik olarak oluşturulup çeşitli otomatik sınama yöntemleri ile birleştirme işlemlerinin sınanmasını sağlamaktadır. Ayrıca bir geliştiricinin kodu sürüm kontrol aracına atması gibi belirli olaylar meydana geldiğinde bu işlemin otomatik olarak yerine getirilmesi sağlanabilmektedir. CruiseControl [11], Apache Continuum [12], Hudson [13] gibi açık kaynak kodlu Java platformu için araçlar bulunmaktadır.

3.6 Otomatik Testler (Automated Tests)

Yazılımların test işlemlerinin elle yapılmasının yavaş ve sıkıcı olması nedeniyle yazılımları otomatik olarak test eden çeşitli araçlar ortaya çıkmıştır. Bu otomatik araçlar bir insan tarafından yapılan testlerin yerine tamamen geçemeseler bile özellikle daha önceden test edilmiş olan mevcut fonksiyonalitenin yazılıma yapılan değişiklikler sonucunda bozulmadığını anlamak için sıklıkla kullanılmaktadır. Büyük projelerde test araçları ile yapılabilecek testleri bu araçlara bırakarak test ekiplerinin bir insan tarafından test edilmesi gereken yeni ve önemli kısımlara yönlendirilebilmesi sağlanabilmektedir. İşlevsel testleri otomatik olarak test eden çeşitli araçların bulunduğu gibi Java platformunda birim testlerin geliştirilmesi ve çalıştırılması için kullanılan açık kaynak kodlu JUnit [14] birim test aracı, otomatik işlevsel testler için bir alt yapı olarak da kullanılabilmektedir.

3.7 Otomatik Yayım

Çok sayıda uygulamanın bir arada çalıştığı ortamlarda yayım işlemlerinin yönetimi zorlaşmaktadır. Özellikle çok katmanlı mimarinin kullanıldığı ortamlarda her katmanda birden fazla sunucunun bulunması dolayısıyla bir katmana yapılacak yayım uygulamanın birden fazla sunucuya yayılmasını gerektirmektedir. Her ne kadar modern işletim sistemleri çeşitli kaynakların etkin kullanımını sağlamış olsalar da çoğu zaman bir sunucuya ait hafıza, aynı anda açılabilen dosya ve thread sayıları, TCP/IP gibi ağ kaynaklarınının daha etkin olarak kullanılması için bir sunucu üzerinde bir uygulamanın birden fazla örneğinin çalıştırılması gerekmektedir. Böyle durumlarda uygulamaların yayılması oldukça zorlaşmaktadır. Otomatik yayım araçları elle yapılan yayım işlemlerinin daha kontrollü ve hızlı olarak yapılabilmesini sağlamaktadır.

3.8 Çalışma Zamanında İzleme

Küçük projelerde yazılımlar çalışma ortamlarına konulduktan sonra genellikle sürekli olarak bakıma veya kontrole gerek duymazlar. Kullanıcılarının az olması veya hizmet kalitelerinin önemli olmaması nedeniyle bu uygulamalar sadece değişiklik gerektiğinde yeniden yayılırlar. Ancak hizmetin kalitesi ve devamlılığı çok önemli olan e-Devlet uygulamaları sürekli olarak bakıma ve kontrole ihtiyaç duyarlar. Aynı performansın ve kalitenin devam ettirilebilmesi için uygulamanın yerine getirdiği işlemlerin sayısı, işlem performansı, çıkan hatalar izlenmelidir. Son zamanlarda ortaya çıkan sistem ve uygulama izleme araçları uygulamaların durumlarının kolaylıkla izlenmesine ve olağan olmayan durumların önceden farkedilebilmesine imkan vermektedir.

4 CyberSoft Application Management Platform (CAMP)

Gelir İdaresi Başkanlığı, Vergi Dairesi Otomasyon Projesi (VEDOP), ismi proje olmasına rağmen aslında program olarak isimlendirilebilecek, içinde Internet üzerinden hizmet veren en büyük e-Devlet projelerinden biri olan E-Beyanname uygulamasının da bulunduğu bir proje grubudur. VEDOP, iç ve dış kullanıcılara hizmet veren çok çeşitli uygulamaları içinde barındırmaktadır. Tüm bu değişik uygulamaların operasyonel isteklerine (iç kullanıcılara hizmet veren projeler mesai saatlerinde kullanılmakta, web uygulamaları 7/24 hizmet vermektedir) cevap verebilmesi, artan kullanıcı taleplerini sorunsuz karşılayabilmesi ve benzeri nedenlerle ölçeklenebilir çok katmanlı mimari kullanılmıştır. Tüm uygulamaların yayıldığı çalışma ortamında 100'e yakın Intel sunucu, 100'ün üzerinde Unix çok işlemcili sunucu bulunmaktadır.

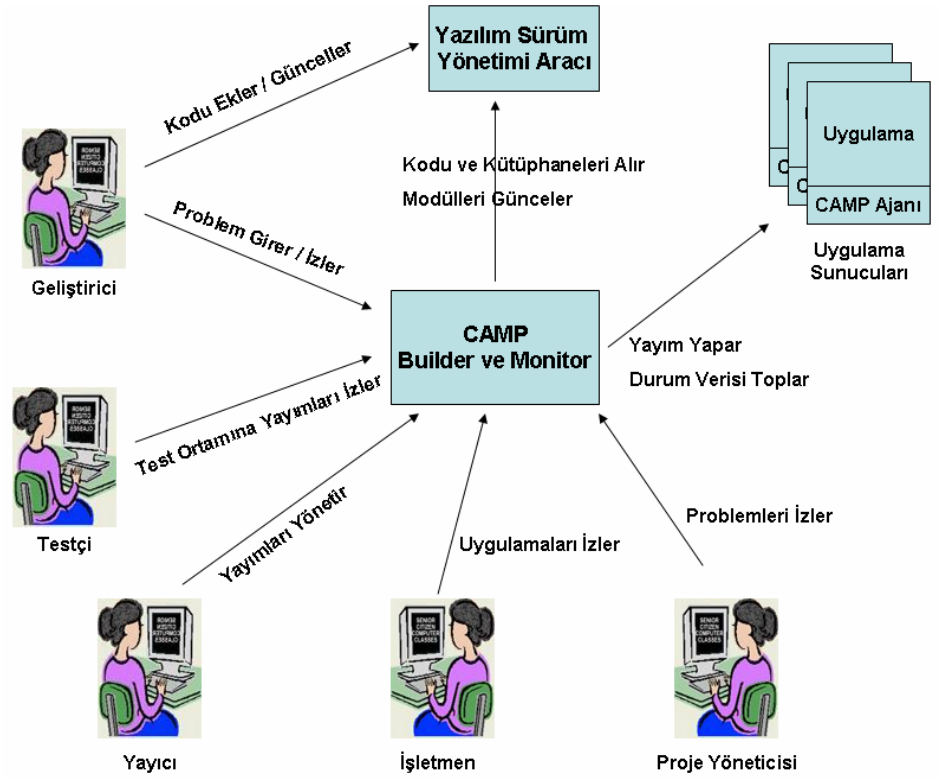
VEDOP içinde Java ve C/C++ gibi çeşitli platformlar, MFC, Swing, HTML, JSP, Servlet gibi çeşitli sunum, RMI, CORBA, Web Service gibi ara katman teknolojileri kullanılmaktadır. Uygulamaların çok büyük sayıda isteğe cevap verebilmesi için her katmanda birden çok sunucu bulunmakta, bu nedenle uygulamalar yerel sunucudaki dosyaları kullanmamak gibi çeşitli kriterler göz önünde bulundurularak belirlenen mimari çerçevesinde geliştirilmektedir.

VEDOP kapsamındaki bu çok sayıdaki değişik uygulamanın yazılım geliştirme ve çalışma ortamlarında izlenebilmesi ve yönetilebilmesi için yukarıda bahsedilen problemleri çözen tekniklerin kullanıldığı bir araca ihtiyaç duyulmuştur. Bu ihtiyaçları karşılamak üzere iki kısımdan oluşan J2EE [17] üzerinde açık kaynak kodlu Ext JS AJAX [16] aracı kullanılarak web arayüzlü CyberSoft Application Management Platform tasarlanmış ve geliştirilmiştir. Şekil 1'de CAMP mimarisi gösterilmiştir.

CAMP Builder. CAMP Builder, uygulamaların çalışacakları ortama yayılmasına kadar olan problem takibi, oluşturma, inceleme, test, yayma aşamalarını tek bir araçla kolaylıkla takip edilebilmesini sağlamaktadır. Geliştirici, test, yayım ekipleri telefon, email gibi klasik iletişim araçlarına gerek kalmadan AJAX tabanlı CAMP Builder arayüzünden yeni problemler girebilmekte ve problemlerin, oluşturma, test, yayım aşamalarından geçişlerini kolaylıkla çevrim içi olarak izleyebilmektedirler. Oluşturma işlemi açık kaynak kodlu Apache Ant [5] oluşturma aracı üzerine kurularak platform bağımsızlığı sağlanmış ve uygulamaların değişik ihtiyaçlarına göre ayarlanabilir ve genişletilebilir bir yapı oluşturulmuştur.

CAMP Monitor. CAMP Monitor, çalışma ortamına yayılan uygulamaların tek bir web tabanlı kontrol panelinden kolaylıkla izlenebilmesi ve yönetilebilmesini sağlamaktadır. Kontrol panelinden modüllerin en son hangi problem numarası ile ne zaman yayımlandığı izlenebilmekte, uygulamalar içindeki sık kullanılan veri ön bellekleri (cache) izlenip yönetilebilmekte, uygulama sunucularının ne zaman kapanıp açıldıkları izlenebilmektedir. İleride uygulamaların yerine getirdikleri işlem sayıları ve işlem performanslarının da izlenebilmesi için bazı eklentilerin yapılması düşünülmektedir. Bu sayede herhangi bir uygulamanın beyanname son beyan tarihlerinde olduğu gibi belirli zamanlarda çok istek alıp performansının azalmaya başladığı rahatlıkla görülebilecektir. Bu durumda CAMP Builder ile oluşturulmuş olan uygulama, çalışma ortamına yeni bir

uygulama sunucusu eklenip CAMP Builder ile kolaylıkla yayılarak uygulamanın hizmet kalitesi hızla eski haline döndürülebilecektir.



Şekil 1. CAMP Mimarisi

5 Sonuç ve Gelecekteki Çalışmalar

Bildiride son zamanlarda yaygınlaşan Uygulama Yaşam Döngüsü Yönetimi kavramı, teknikleri, araçları ve faydaları incelenmiştir. Bahsedilen araç ve tekniklerin kullanımı ile proje ekipleri arasındaki iletişim ve koordinasyon artmakta, işlerin izlenmesi ve yönetilmesi kolaylaşmaktadır.

Gelir İdaresi Başkanlığı'nın Vergi Dairesi Otomasyon Projesi (VEDOP) kapsamındaki uygulamaların geliştirilmesi ve yönetilmesi aşamalarının daha kolaylaştırılması için geliştirilen CyberSoft Application Management Platform (CAMP) incelenmiş ve çalışma prensipleri ve mimarisi gösterilmiştir.

Uygulama Yaşam Döngüsü Yönetimi Teknikleri başlıklı 3. bölümde her bir aşama/teknik için geliştirilmiş olan çeşitli araçlar listelenmiştir. Bu araçlar 2. bölümde

de bahsedildiği gibi tek başlarına çözmek için tasarlandıkları problemleri gayet iyi bir şekilde çözmelerine rağmen sadece planlanan kullanım alanı içinde kalmakta, farklı aşamalar/tekniklerle tam bir entegrasyona olanak sağlamamaktadır. CAMP ise, bu araçların yerine getirdikleri işlevleri tek bir ortam altında birleştirmektedir. Bu araçlardan bazıları (statik kod incelemesi araçları, birim test araçları, vb) özel bir değişikliğe gerek kalmadan sistem ile tam olarak entegre olabilmekte, bazıları ise (yazılım konfigürasyon yönetimi araçları, vb) CAMP içine konulan özel modüller ile sisteme entegre edilmektedirler.

İleride CAMP'e uygulamaların performanslarının izlenebileceği bir eklenti konularak çalışma ortamında performans yavaşlamalarının daha hızlı olarak farkedilebilmesi sağlanabilir. Eğer mümkün olursa CAMP'in otomatik olarak az yoğun olan bir sunucu üzerinde yavaşlayan uygulamanın yeni bir örneğini daha açarak insan müdahalesine gerek kalmadan veya çok aza indirilerek uygulamaların yoğun zamanlarda hizmet kalitelerinin devamlılığı sağlanabilir.

Kaynakça

1. Şelale Modeli, http://en.wikipedia.org/wiki/Waterfall_model
2. Tekrarlı Süreç Modeli, http://en.wikipedia.org/wiki/ Iterative_and_incremental_development
3. Çevik Yazılım Geliştirme Bildirisi, http://www.agilemanifesto.org/
4. Uygulama Yaşam Döngüsü Yönetimi, http://en.wikipedia.org/wiki/Application_Lifecycle_Management
5. Apache Ant, http://ant.apache.org/
6. Statik Kod İncelemesi, http://en.wikipedia.org/wiki/Static_code_analysis
7. Checkstyle, http://checkstyle.sourceforge.net/
8. PMD, http://pmd.sourceforge.net/
9. FindBugs, http://findbugs.sourceforge.net/
10. Sürekli Birleştirme, Martin Fowler, http://www.martinfowler.com/articles/continuousIntegration.html
11. CruiseControl, http://cruisecontrol.sourceforge.net/
12. Apache Continuum, http://continuum.apache.org/
13. Hudson, https://hudson.dev.java.net/
14. JUnit, http://www.junit.org/
15. Carey Schwaber, John R. Rymer, Jacqueline Stone, The Changing Face Of Application Life-Cycle Management, Ağustos 2006, http://www.forrester.com/Research/Document/Excerpt/0,7211,37653,00.html
16. Ext JS Javascript Library, http://extjs.com/
17. Java 2 Platform Enterprise Edition, http://java.sun.com/j2ee/overview.html
18. Java, http://www.sun.com/java/

Anlamsal Web Ortamında Çalışacak Çok-Etmenli Sistemler için bir Referans Mimarisi

Geylani Kardaş[1], Oğuz Dikenelli[2]

[1] Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, 35100, Bornova, İzmir
[2] Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, 35100, Bornova, İzmir
[1] geylani.kardas@ege.edu.tr, [2] oguz.dikenelli@ege.edu.tr

Özet. Yazılım etmenleri ve bunların oluşturduğu çok-etmenli sistemler (ÇES), karmaşık yapıdaki dağıtık sistemlerin modellenmesini ve oluşturulmasını sağlayan etkili birer teknoloji olarak ortaya çıkmışlardır. Anlamsal Web evrimi de şüphesiz etmen araştırmalarına yeni bir vizyon getirmiştir. Bu ikinci nesil Web, web sayfası içeriklerini, ontolojiler kullanılarak yorumlanabilecek bir seviyeye taşımayı hedeflemektedir. Söz konusu yorumlamanın ve anlam çıkarsamaların özerk etmenler tarafından insanlar adına yerine getirileceği düşünülmektedir. Özellikle anlamsal web servisi gibi Anlamsal Web yapıları ile yazılım etmenlerinin etkileşiminin ÇES'ler tasarlanıp uygulamaya geçirilirken dikkate alınması ve sistem mimarilerinin de ilgili ihtiyaçlara cevap verecek bileşenlere sahip olması gerekmektedir. Bu bildiride Anlamsal Web ortamında çalışacak ÇES'ler için bu amaca yönelik katmanlı bir referans mimarisi önerilmektedir. Mimari etmenlerin iletişim, içsel planlama mekanizmaları ve bağlı bulundukları platform bünyesinde sundukları veya hizmet aldıkları servislere yönelik sistem bileşenlerini tanımlamakta ve ilişkilerini ortaya koymaktadır.

1 Giriş

Yazılım etmenleri (*"software agents"*) kullanıcılarının adına bir takım görevleri yerine getirmek üzere davranma yeteneği olan özerk (*"autonomous"*) yazılım bileşenleri olarak tanımlanmaktadır. Öte yandan bir çok akıllı yazılım etmeninin bir araya gelerek oluşturdukları ve kendi bilgi ve bireysel yeteneklerini kullanarak çözemedikleri veya etkin bir biçimde çözemeyeceklerini düşündükleri problemlerini çözmek amacıyla birbirleri ile etkileşimde bulundukları sistemler ise Çok-etmenli Sistem (*"Multi-agent System"*) (ÇES) adını almaktadır [1]. Söz konusu etmen etkileşimleri [2]'de belirtildiği gibi bencil veya işbirlikçi bir yapıda olabilir. Başka bir deyişle etmenler ortak bir amacı paylaşabilir ya da serbest piyasa ekonomisinde olduğu gibi kendi çıkarlarının takipçisi olabilirler.

Weyns ve Holvoet, [3]'te bir ÇES'in belirli bir problemi çözmek için gerekli yazılımı sağladığını belirtmektedirler. Bu amaç doğrultusunda ÇES ilgili sistemi birbirleriyle etkileşim halinde olan bir dizi özerk varlıklar halinde yapılandırmakta ve sistemin işlevi ve kalitesine yönelik ihtiyaçları karşılamaktadır. Öte yandan [4]'te bir *yazılım mimarisi*, yazılım elemanlarını, bu elemanların görünür özelliklerini ve elemanlar arasındaki ilişkileri içeren bir sistemin yapı veya yapıları olarak tanımlanmaktadır. Yazılım elemanları (ya da daha genel bir deyişle mimariye ait elemanlar) sistemin işlevselliğini sağlarken ihtiyaç duyulan sistem kalite özellikleri yazılım mimarisinin yapıları

üzerinden karşılanmaktadır. [3]'teki bakış açısı dikkate alındığında ÇES'ler ile yazılım mimarisi arasında çok yakın bir ilişkinin olduğu söylenebilir. Çünkü bir ÇES asıl hedefini yerine getirmek için ne yapıyorsa bir yazılım mimarisi de onu içermektedir.

Literatürde ÇES'lerin mimari özelliklerini göz önüne alan çalışmaların yanında ÇES mimarilerine organizasyonel perspektifte, ilgi yönelimli (*"aspect-oriented"*) veya model tabanlı öneriler sunan çeşitli çalışmalar (örneğin [5], [6], [7], [8], [9] ve [10]) bulunmaktadır. Her ne kadar söz konusu bu çalışmalar ilgili alana önemli katkılar sağlamış olsa da yakın gelecekte etmenlerin üzerinde çalışacağı düşünülen Anlamsal Web (*"Semantic Web"*) ortamı ve ÇES'lerin bu ortam üzerinde çalışabilmesi için ihtiyaç duyulan yapıların bu çalışmalarda desteklenmediği gözlenmiştir.

Anlamsal Web evrimi [11] şüphesiz etmen araştırmalarına yeni bir vizyon getirmiştir. Bu ikinci nesil Web, Dünya Geneli Ağ'ı (WWW) web sayfası içeriklerinin ontolojiler kullanılarak yorumlanabileceği bir seviyeye taşımayı hedeflemektedir. Söz konusu yorumlamanın ve anlam çıkarsamaların özerk etmenler tarafından insanlar adına yerine getirileceği düşünülmektedir.

Anlamsal Web ortamının kendine özgü mimari varlıklarının ve farklı bir anlamsal yapısının olduğu, bu ortam üzerinde çalışacak ÇES'ler hazırlanırken göz önünde bulundurulmalıdır. Etmen mimarilerinin, modelleme tekniklerinin ve ÇES yazılımı geliştirme çerçevelerinin bu yeni ortamı desteklemesi gerekmektedir. Bu düşünceden hareketle bu bildiride Anlamsal Web ortamında çalışacak ÇES'ler için bir referans mimari tanıtılmaktadır. Önerilen mimari, bünyesinde bir etmenin Anlamsal Web ortamında hem diğer etmenler hem de Anlamsal Web ortamına özgü anlamsal web servisleri ile etkileşimine ait servis, etmen planlama ve iletişim seviyesinde görev alan mimari bileşenlerini içermektedir. Böyle bir mimariye dayalı olarak hayata geçirilen ÇES'ler Anlamsal Web yetenekli olacaklar ve bu ÇES'lerde yer alan yazılım etmenleri de kullanıcıları adına Web içeriğini farklı kaynaklardan elde edebilecek, bilgiyi işleyebilecek ve sonuçları değiş tokuş edebileceklerdir. Ayrıca özerk etmenler bu tip ÇES'ler içerisinde anlamsal veriyi değerlendirebilecek ve içerik dilleri vasıtasıyla anlamsal web servisleri gibi anlamsal ortam elemanları ile etkileşimlerde bulunabileceklerdir.

Bir referans mimarisi, benzer özelliklere ve sistem ihtiyaçlarına sahip bir dizi uygulamanın tasarımı ve geliştirilmesi sırasında elde edilen deneyimler sonucunda şekillenmektedir [12] ve sunduğu ortak zemini paylaşan yeni yazılım mimarilerinin geliştirilmesinde yazılım mimarlarına fayda sağlamaktadır. Bildiride tanıtılan mimarinin de bu amaç doğrultusunda Anlamsal Web yetenekli ÇES'ler için somut mimariler hazırlanırken kullanılabilecek bir referans mimarisi olacağına inanılmaktadır.

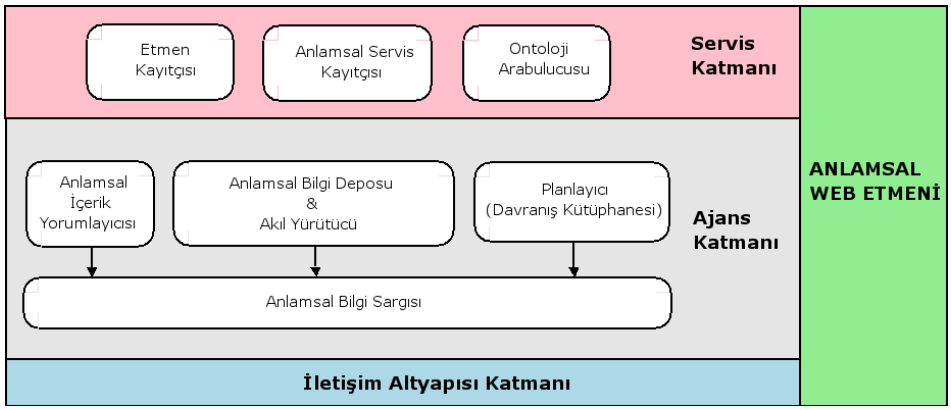
Bildirinin geriye kalan kısmı şu şekilde düzenlenmiştir: Bölüm 2'de Anlamsal Web ortamında çalışacak ÇES'lere ait referans mimarisi anlatılmaktadır. Bölüm 3'te ÇES'ler için yazılım mimarisi öneren literatürdeki diğer çalışmalar özetlenmiştir. Elde edilen sonuçlar ve hedeflenen ileriye yönelik çalışmalar ise Bölüm 4'te yer almaktadır.

2 Anlamsal Web Etmenleri için Referans Mimarisi

Anlamsal Web'in gerçek gücü Web içeriğini farklı kaynaklardan toplayabilen, elde ettiği bilgileri işleyebilen ve çıkarsadığı sonuçları başka ortam elemanları ile de paylaşabilen bilgisayar programlarının geliştirilmesiyle ortaya çıkacaktır [11]. Söz konusu bu bilgisayar programları yazılım etmenleridir. Yazılım etmenlerinin etkinliği makineler tarafından anlaşılabilen Web içeriği daha fazla hazır hale geldikçe ve kullanılabilir, otomatikleştirilmiş (*"automated"*) servislerin sayısı çoğaldıkça artacaktır.

Yukarıda sözü edilen ortamda çalışacak etmen yazılımlarının tasarlanması ve kullanılmasına yönelik bir yazılım mimarisinin ortaya konulması gerekmektedir [13]. Bu amaç doğrultusunda önerdiğimiz referans mimari Şekil 1'de verilmiştir.

Önerilen referans mimari, [14]'te de anlatılan *Modül Bakış Tipi* (*"Module Viewtype"*) göz önüne alınarak hazırlanmıştır ve katmanlı bir yapıya sahiptir. Katmanlar arası ilişki *kullanıma izinli* (*"allowed-to-use"*) adı verilen ilişki gösterim tipindedir. Clements ve ark.'nın [14]'te verdikleri tanıma göre aralarında bu ilişki olan iki katmandan ilkinde yer alan herhangi bir modül ikinci katmanda yer alan herhangi bir modülü kullanma hakkına sahiptir. İlişkinin yönü aşağıya doğrudur. Bunun anlamı önerilen mimaride sadece üst seviye bir katman alt seviyedeki bir katmanın sunmuş olduğu servis veya hizmetleri kullanabilir. Tersine izin verilmemektedir. Öte yandan önerilen mimari herhangi bir *katman köprüleme*yi (*"layer bridging"*) [14] de içermemektedir. Buna göre bir üst katman sadece bir sonraki alt katmanın modüllerini kullanabilir. Mimariyi resimlemek için kullandığımız gösterim Şekil 1'de görüldüğü üzere yan eklentili katmanları (*"layers with a sidecar"*) [14] içermektedir. Katmanlar arası *kullanıma izinli* ilişkisi ise şekilde geometrik komşuluklarla temsil edilmektedir.



Şekil 1. Anlamsal Web ortamında çalışan ÇES'ler için katmanlı bir referans mimarisi

Önerilen mimarinin ana kısmında üç adet katman tanımlanmıştır: *Servis Katmanı*, *Ajans* (*"Agency"*) *Katmanı* ve *İletişim Altyapısı Katmanı*. *Anlamsal Web Etmeni* ise bu

mimari katmanlarında bulunan tüm modülleri (bileşenleri) kullanma hakkına sahip mimarinin eklenti (*"sidecar"*) bileşenidir. Bu mimariye uygun olarak geliştirilen bir ÇES'te bir grup etmen Servis Katmanı'nda tanımlı servisleri sunar. Sistemdeki her etmenin ise Ajans Katmanı'nda tanımlı bir etmen iç yapısı bulunmaktadır. Sistemdeki etmenler birbirleri ile İletişim Altyapısı Katmanı'nda tanımlanmış olan protokollere uygun olarak haberleşirler. Söz konusu bu mimari katmanları aşağıdaki alt bölümlerde detaylı olarak anlatılmaktadır.

2.1 Servis Katmanı

Servis Katmanı'nda bir ÇES'te yer alan anlamsal web etmenlerinin servisleri (ve/veya rolleri) tanımlanmaktadır. Servis Katmanı'ndaki tüm servisler Ajans Katmanı'nın sunmuş olduğu imkanları kullanırlar. İlgili etmen sisteminin iş alanına özgü etmen servisleri haricinde bu katmanda, etmen sistemine sağlanması gereken sarı sayfa ve arabulucu servisleri yer almaktadır.

Etmen Kayıtçısı, etmen platformunun diğer üyeleri için sistemde yer alan etmenlerin yeteneklerinin anlamsal olarak tanımlandığı ve ilan edildiği bir sistem servisidir. Görevlerinin işletimi sırasında platform etmenleri diğer etmenler tarafından sunulan servislere ihtiyaç duyabilir. Bu nedenle söz konusu bu servis üzerinde sorgular işletirler ve etkileşim için uygun etmenleri belirlerler.

Geleneksel bir ÇES, sistemin üyesi olan etmenlerin uygun etmen servislerini bulabilmesi amacıyla sarı sayfa hizmeti sunan bir ya da daha fazla kayıtçıya sahiptir. Örneğin FIPA[1] soyut mimari tanımında etmenlerin sunduğu servislerin kayıt olduğu, *Dizin Kolaylaştırıcısı (DK)* adı verilen ve her FIPA uyumlu ÇES'te olması gereken bir etmen çeşidi bulunmaktadır [15]. Bir etmen spesifik bir etmen servisini aradığında DK'dan servisi sağlayan etmenin bilgilerini (örneğin etmenin adı, adresi, vb.) elde etmekte ve görevini tamamlamak üzere ilgili servisi sağlayan etmenle iletişime geçmektedir.

FIPA uyumlu olsun ya da olmasın etkileşim içerisindeki etmenleri içeren bir ÇES'te yukarıda tarif edilen etmen kayıtçılarının olması gerekliliği açıktır. Ancak etmenlerin talep ettikleri ve sundukları servislerin (bir anlamda etmen yeteneklerinin) eşlenmesi işlemi Anlamsal Web ortamında çalışacak ÇES'ler düşünüldüğünde daha karmaşık bir yapıya bürünmektedir ve yeniden tanımlamaya ihtiyaç duymaktadır. Bu tip ÇES'lerde etmen servislerinin keşfi için servis yeteneklerinin anlamsal eşlenmesine ait kriterlerin tanımlanması ve etmen servisi tanımlarının kaydedilme mekanizmalarının (bir anlamda dizin servislerinin) bu kriterlere uygun olarak tasarlanması gerekmektedir. Böylelikle aranan servis özellikleri ve ilan edilen servislerin yetenekleri arasındaki eşleme işlemi sadece özdeş (*"identical"*) servis eşlemeyi göz önünde bulundurmayarak daha etkin bir hale gelmektedir. Buradaki özdeş servis eşleme ile kastedilen standart dizin servislerinde yer alan anahtar kelime bazlı servis arama ve eşleme işlemidir. Oysa yeni

[1] FIPA ("Foundation for Intelligent Physical Agents"), etmenler ve etmen tabanlı sistemler arasında birlikte çalışabilirliği desteklemek amacıyla çeşitli standartlar ortaya koyan bir kuruluştur. Bu standartlara dayalı olarak tanımlanmış olan FIPA ÇES platformu literatürde üzerinde en çok çalışılan ve desteklenen ÇES platformudur. FIPA ÇES platformu ve FIPA standartları hakkında ayrıntılı bilgiye http://www.fipa.org adresinden erişilebilinir.

yetenek eşleme süreci aranan ve ilan edilen iki servis arasındaki ilişkinin tipini ve derecesini *anlamsal olarak* belirleyecektir ve bu da etmen ihtiyaçlarını karşılayan en uygun servislerin bulunmasını sağlayacaktır. Tüm bu vizyona uygun olarak referans mimarisi etmen servisleri üzerinde yetenek eşlemesini yerine getirecek bir etmen kayıtçısını Servis Katmanı'nda tanımlamaktadır. Anlamsal yetenek eşlemesi ile ilgili detaylar bu bildirinin kapsamı dışındadır ancak FIPA uyumlu etmen sistemleri için örnek bir anlamsal yetenek eşleme mekanizması [16]'da anlatılmaktadır.

Öte yandan bir ÇES'te yer alan Anlamsal Web yetenekli yazılım etmenleri, yerine getirmek istedikleri görevlerinin işletimi sırasında eğer ihtiyaç hissederlerse *anlamsal web servisleri* ile de etkileşime geçebilirler. Anlamsal web servisleri, bulunmaları ve otomatik olarak işletilmeleri için anlamsal bir ara yüze sahip web servisleri olarak tanımlanabilirler.

Günümüzde kullanılmakta olan web servisleri kendilerini temsil eden ve *Web Servisleri Tanımlama Dili* (*"Web Services Description Language - WSDL"*) kullanılarak hazırlanan ara yüzleri sayesinde geliştirildikleri yazılım dili ve/veya ortamına bağlı kalmaksızın yine çok çeşitli ortamlarda çalışan istemci yazılımlar tarafından kullanılabilmektedirler. Bir istemci program WSDL'i işleyerek, sunulan servise ait işlemi ve bu işlem için gerekli girdi – çıktı parametrelerini öğrenir ve servisi WSDL gibi yine web servisleri için bir standart olan *Basit Nesne Erişim Protokolü* (*"Simple Object Access Protocol - SOAP"*) kullanarak çalıştırabilir. Ancak bu mevcut web servis altyapısı sadece sözdizimsel birlikte işlerliği göz önünde tutmaktadır ve [17]'de belirtildiği gibi böyle bir yaklaşım ne anlamsal birlikte işlerliği ne de web servislerinin otomatik tümleşimini mümkün kılar. Söz konusu birlikte işlerliği ve tümleşimi sağlamak amacıyla web servislerinin yeteneklerinin servis ontolojilerinde tutulması ve bu ontolojiler kullanılarak ihtiyaca en uygun servislerin bulunmasına ve dinamik çağrımının gerçekleştirilmesine çalışılmaktadır.

Servis yeteneklerinin tanımlanması ve servis çalıştırma sürecinin ifade edilmesine yönelik *Servisler için Web Ontoloji Dili* (*"OWL for Services - OWL-S"*) [18] ve *Web Servis Modelleme Ontolojisi* (*"Web Service Modeling Ontology - WSMO"*) [19] gibi çeşitli anlamsal web servis tanımlama ve kullanma ontolojileri literatürde bulunmaktadır. Bizim bakış açımıza göre, bu şekilde tanımlanmış servis yeteneklerinin de tıpkı etmen servisleri için olduğu gibi uygun kayıtçılarda tutulup ilan edilmesi gerekmektedir. Böylelikle bu servisler de etmenler tarafından dinamik olarak keşfedilecek ve ihtiyaçları doğrultusunda çalıştırılabileceklerdir. Önerdiğimiz mimaride bu amaca uygun olarak *Anlamsal Servis Kayıtçısı* adı verilen bir Servis Katmanı bileşeni tanımlanmıştır.

Anlamsal Servis Kayıtçısı, ilgili platforma ait anlamsal web servislerinin ara yüzlerini bu servislerin etmenler tarafından keşfedilmesi amacıyla ilan eden bir servis eşleyici olarak modellenebilir. Örneğin OWL-S servislerini göz önüne alacak olursak, bir etmen ihtiyaç duyduğu anlamsal servisin yeteneklerini belirten OWL-S profilini bu kayıtçıya göndererek bu kayıtçı üzerinde ilgili sorgunun işletilmesini sağlar. Anlamsal Servis Kayıtçısı verilen ihtiyaç profili ve ilan ettiği servis profilleri arasında bir anlamsal yetenek eşleme işlemi gerçekleştirerek etmenin işine yarayacak uygun servisleri ilgili etmene bildirir. Etmen de uygun olan bu servis (ya da servislerle) anlaştıktan sonra görevini tamamlamak üzere ilgili servis (ya da servislerle) etkileşimde bulunabilir. Söz

konusu anlaşma ve servis çalıştırma işlemi yine anlamsal olarak tanımlanmış etkileşim protokollerine uygun olarak yerine getirilmektedir. Anlamsal web servisleri için yetenek eşleme hakkında detaylı bilgiye ve örnek bir eşleme mekanizmasına [20}'den erişilebilir. Anlamsal web servislerinin ÇES'ler bünyesinde kullanımı ve yukarıda sözü edilen Anlamsal Servis Kayıtçısı'nın somut bir uygulaması da [21]'de anlatılmıştır.

Hem Etmen Kayıtçısı hem de Anlamsal Servis Kayıtçısı bir ÇES'te servis eşleyici yerine birer aracı (*"broker"*) olarak da uygulamaya geçirilebilirler. Bu durumda ilgili kayıtçılar sadece servis yetenek eşlemesini gerçekleştirmezler; buna ek olarak servise ihtiyaç duyan etmen adına servisle bizzat etkileşime geçerler ve servis çalıştırma sonucunu servisi talep eden etmene yönlendirirler.

Bir anlamsal web etmeni farklı etmen organizasyonlarındaki etmenlerle etkileşimde bulunma ihtiyacı hissedebilir. Ayrıca bu etmenlerin ve anlamsal web servislerinin farklı bilgi depolarında ya da dağıtık sistemlerde yer alan bilgi kaynaklarını kullanması gerekebilir. ÇES'ler gibi açık sistemlerde bu durumlar nedeniyle birden fazla ontoloji yer alabilir ve farklı sistem elemanları farklı ontolojileri kullanabilir. Bu nedenle farklı ontolojilerin kavram dönüşümlerini ve eşlemelerini yerine getirecek servislerin ÇES'lerde olması gerektiğine inanmaktayız. Şekil 1'de görüldüğü gibi referans mimaride bu servisi sağlayan bileşene *Ontoloji Arabulucusu* adı verilmiştir. Bir ÇES bünyesinde bir ya da daha fazla Ontoloji Arabulucusu yer alabilir.

Bir Ontoloji Arabulucusu aynı zamanda ÇES iş alanına ait ve ilgili platform içerisindeki elemanlarca kullanılan ontolojiler için merkezi bir veri havuzu görevini üstlenebilir ve ontoloji yükleme, ontoloji güncelleme ve ontolojiler üzerinde sorgu gerçekleştirme gibi temel ontoloji yönetim işlemlerini yürütebilir. Ontoloji Arabulucusu servisini sağlayan bir etmen ontoloji çevrim isteklerini özel bir kullanıcı ara yüzü vasıtası ile önceden tanımlanmış kavram eşleme bilgisine göre karşılamaktadır. Sağlanan bu ontoloji çevrim desteği ile bir etmen kendisinin üyesi olmadığı başka bir ÇES'te yer alan bir etmenle ya da kendi ortamı dışında yer alan bir servis ile farklı ontolojiler kullansalar bile iletişimde bulunabilir.

2.2 Ajans Katmanı

Referans mimarinin orta katmanı olan Ajans Katmanı anlamsal web etmenlerinin iç yapısını tanımlamaktadır. Özerk ve karşıt eylemli (*"reactive"*) yapıdaki etmenlerin amaçlarına uygun olarak dış ortam bileşenlerini nasıl kullanacağı ve davranışları için nasıl plan yapacağı bu iç yapıya göre belirlenmektedir.

Sistemdeki her etmenin yerel ontolojilerini sakladığı bir *Anlamsal Bilgi Deposu* bulunmaktadır. Bu ontolojiler etmen platformundaki diğer etmenler veya servisler ile etkileşime geçerken kullanılmaktadır. Ontolojilerin değerlendirilmesi ve temel çıkarsama ise *Akıl Yürütücü* modülü tarafından yerine getirilir.

Anlamsal İçerik Yorumlayıcısı etmen iletişimini kontrol eder. Bir anlamsal web etmeni iletişimleri sırasında diğer etmenlerden veya anlamsal servislerden doğal olarak mesajlar alacaktır. Alınan mesaj içeriğinin anlamsal uygunluğunun kontrol edilmesine ve içeriğin etmenin inançlarına ve niyetlerine uygun bir şekilde yorumlanmasına ihtiyaç vardır. Anlamsal İçerik Yorumlayıcısı söz konusu bu içerik uygunluğunun kontrolünü ve yorumlamayı gerçekleştirmektedir.

Ajans Katmanı'nın *Planlayıcı* adı verilen modülü ihtiyaç duyulan yeniden kullanılabilir etmen planlarını ve ilgili davranış kütüphanelerini içermektedir. Yeniden kullanılabilir etmen planları bir etmenin niyetlerine uygun olarak işletilen görevlerinin birleşiminden oluşmaktadır. Planlayıcı, örneğin Hiyerarşik Görev Ağı (*"Hierarchical Task Network – HTN"*) [22] gibi bir karşıt eylemli planlama (*"reactive planning"*) paradigmasını temel almaktadır. Dickinson ve Wooldridge'in [23]'te belirttiği gibi karşıt eylemli planlama için bir etmene önceden tanımlı (belki de sistemin derlenme zamanında tanımlanmış) genel planlardan oluşan bir kütüphane sağlanmaktadır. Etmen ortamdan elde ettiği algılara tepki olarak bu planların birini ya da birkaçını uygulamaya geçirir.

Öte yandan bir yapay zeka planlama metodolojisi olan HTN planlama, görevlerin ayrıştırılması (*"decomposition"*) ilkesine bağlı olarak planların ortaya konmasını sağlar. Bu ilkeye göre doğrudan çalıştırılabilen görevlerin oluşturduğu, hiyerarşide en üstte yer alan ana bir plan vardır. Planlama sistemi doğrudan çalıştırılabilecek en temel görevleri bulana kadar bu plan ayrıştırılmakta ve etmen bu görevleri yerine getirmektedir. Örneğin bir etmenin anlamsal web servisleri ile etkileşimi servis keşfi, servisle anlaşma ve servisi çalıştırma görevlerinin birleşiminden oluşan yeniden kullanılabilir bir plan olarak modellenebilir. Anlamsal web servisini kullanabilmek için etmen bu görevler kapsamında tanımlanan işlemleri yerine getirir.

Ajans Katmanı'nda bulunan *Anlamsal Bilgi Sargısı* (*"Semantic Knowledge Wrapper"*) yukarıda sözü edilen ontolojilerin Ajans Katmanı'nın üst seviye bileşenleri tarafından kullanılabilmesini sağlamaktadır. Örneğin görev işletimi sırasında bir etmen bir ontoloji varlığının nesne (ya da başka bir programlanabilir yapı) gösterimine ihtiyaç duyabilir. Anlamsal İçerik Yorumlayıcısı da etmen ontolojileri üzerinde sorgu işleterek bir konu hakkında çıkarsamada bulunmak isteyebilir. Bu tip ihtiyaçları gidermek amacıyla Anlamsal Bilgi Sargısı, Ajans Katmanı'nın çalışma zamanı ortamı içerisinde ilgili ontolojilerin çizge gösterimlerini oluşturabilir. Etmen iç mimarisinde böyle bir sargının kullanılmasına dair bir örnek JENA[1] çerçevesine dayalıdır ve [24]'te anlatılmıştır.

2.3 İletişim Altyapısı Katmanı

Mimarinin en alt katmanı mimarinin iletişim altyapısı uygulamasının soyutlanmasından sorumludur. Referans mimarisinin somut bir örneğinde bu katman FIPA Etmen İletişim ve Etmen Mesaj Taşıma protokollerinin bir uygulaması olabilir. Böylece altyapı FIPA Etmen İletişlim Dili'nin ("Agent Commnication Langauge - ACL") ve ilgili protokolün kullanılması ile etmenler arası mesaj transferini gerçekleştirmiş olur. Fiziksel iletim iyi bilinen HTTP-IIOP (*"HTTP - Internet Inter-ORB Protocol"*) üzerinden gerçekleştirilebilir. Burada asıl önem verilmesi gereken mesaj altyapısında kullanılan içerik dilinin yapısı ve zenginliğidir.

[1] JENA, Anlamsal Web uygulamalarının geliştirilmesini sağlayan ve Java programlama dili kullanılarak hazırlanmış açık kaynak kodlu bir çerçevedir. Ontolojilerin kullanılması için programlanabilir bir ortam sağlamaktadır ve kural tabanlı bir çıkarsama motoru içermektedir. Tüm çerçeveye http://jena.sourceforge.net/ adresinden erişilebilinir.

3 İlgili Çalışmalar

Etmen araştırmacılarının literatürde ÇES'ler için önerdikleri mimari çalışmalarının farklı bakış açılarına sahip olduğu gözlenmektedir. [5]'teki çalışmada Shehory, ÇES'lerin yazılım mühendisliğindeki rollerini değerlendirmekte ve tasarımcıların bir problemin çözümü için önerilen bir ÇES'in uygunluğunu değerlendirmede kullanabilecekleri başlangıç seviyesindeki mimari özellikleri sunmaktadır. Sunulan bu özellikler ÇES'leri bir yazılım mimarisi stili olarak karakterize etmeyi sağlamaktadırlar. Öte yandan ÇES mimarilerine organizasyonel perspektiften bakan [6]'daki çalışmada organizasyon yönetimi teorisi kavramlarını içeren bir dizi mimari stili önerilmiştir. İlgili ÇES mimari stilleri aktör, görev ve aktör bağımlılığı gibi ÇES kavramlarını ön planda tutan bir çerçeveye göre modellenmişlerdir.

[25]'te tanıtılan PROSA, imalat sistemlerini göz önüne alan koordinasyon ve kontrol uygulamaları için bir referans mimarisidir. Kaynak etmeni, ürün etmeni ve sipariş etmeni adı verilen üç temel etmen tipi üzerine kurulu PROSA'da etmenler ve aralarındaki ilişkileri belirlemede nesne yönelimli kavramlar kullanılmıştır. Weyns ve Holvoet de [7]'de yerleşik (*"situated"*) ÇES'ler için bir referans mimarisi önermişlerdir. Söz konusu mimari ilgili araştırmacıların üzerinde çalıştıkları çeşitli robotik uygulamalarının ortak fonksiyonları ve yapıları belirlenerek oluşturulmuştur ve bu uygulamalar sonucunda elde edilen deneyimi yansıtmaktadır.

FIPA'nın ÇES'ler için önerdiği FIPA Soyut Mimarisi [15] farklı ileti taşıma protokolleri, farklı etmen iletişim dilleri ve farklı içerik dilleri kullanan etmenler arasında anlamlı ileti alışverişini sağlamayı amaçlamaktadır. Önerilen soyut mimari, somut mimarilerin geliştirilmesinde temel olarak alınmaktadır. Bir somut mimarinin FIPA uyumlu olabilmesi için, etmenleri kaydeden, etmenleri bulan ve etmenler arası ileti transferini gerçekleştiren mekanizmalara sahip olması gerekmektedir.

Garcia ve ark., [8]'de özerklik, öğrenme ve taşınabilirlik gibi etmen özelliklerinin örneğin [9]'da olduğu gibi klasik mimari desenlerini uygulayan yaklaşımlarla karşılanamadığını savunmuş ve mimarilerin yeniden kullanılabilirliğinin ve yönetiminin varolan bu yaklaşımlarda oldukça zor olduğunu belirtmişlerdir. Bu nedenle etmen mimarilerini yapılandırmak için ilgi yönelimli bir yaklaşım önermişlerdir.

ÇES'ler için üstmodeller tanımlayıp bu modeller arası dönüşümler sonrası etmen yazılımlarını elde etmeyi hedefleyen model güdümlü etmen mimari çalışmaları da bu alanda öneme sahiptir. [10], [26] ve [27] gibi önerilerde ortam bağımsız ya da belli bir ortama uygun ÇES'lerin model güdümlü olarak geliştirilmesi amaçlanmaktadır.

Bildirinin giriş bölümünde de belirtildiği gibi yukarıda değinilen bu önemli etmen mimarisi çalışmalarında Anlamsal Web ortamı ve ÇES'lerin bu ortam üzerinde çalışabilmesi için ihtiyaç duyulan yapıların desteklenmediği görülmektedir. Önerilen referans mimarisinin çalışmalarda gözlemlenen bu eksikliği kapamaya yönelik bir ilk adımı temsil ettiği söylenebilir.

4 Sonuç ve İleriye Yönelik Çalışmalar

Bu bildiride anlatılan referans mimarisi Anlamsal Web ortamında çalışacak etmen sistemlerinin tasarımı ve uygulaması için gerekli yapıları tanımlamayı hedeflemektedir. Mimarinin öncül bir versiyonu [13]'teki çalışmada da yer almaktadır. Ancak bu öncül versiyonda katmanlar arası ilişkiler belirgin değildir ve gerek katmanların gerekse de katmanlar içerisindeki modüllerin gösteriminde ihtiyaç duyulan biçimsellik eksiktir. Bu bildiride anlatılan çalışma ile mimarinin söz konusu ihtiyaçlar doğrultusunda yenilenmesi yerine getirilmiştir.

Önerilen referans mimari Bölüm 2'de de belirtildiği gibi, yazılım mimarilerini belgelemek için [14]'te yer alan bakış tiplerinden *Modül Bakış Tipi* (*"Module Viewtype"*) göz önüne alınarak hazırlanmıştır ve gereksinimleri karşılayacağına inanılan ilgili modül bakış tipi stili (*katmanlı stil*) ile bu çalışmada anlatılmıştır. Ancak referans mimarisinin, mimariyi farklı perspektiflerden tanımlayan diğer mimari bakış tipleri ile de desteklenmesi gerekmektedir. Bu amaçla yakın gelecekte, Anlamsal Web yetenekli ÇES'lerde yer alan etmenlerin çalışma zamanı davranışlarını ifade eden stilleri içerdiği düşünülen *Bileşen ve Bağlaç Bakış Tipi* (*"Component-and-Connector Viewtype"*) [14] göz önüne alınarak mimari başka bir perspektiften değerlendirilecek ve belgelendirilecektir. Bir sonraki adım da ise yine bu tip ÇES'lere ait bileşenlerin üstlendikleri görevler doğrultusunda sistem içerisindeki konumlarını belirlemek amacıyla *Yerleşim Bakış Tipi* (*"Allocation Viewtype"*) [14] stillerinin incelenmesi ve uygun olan veya olanlarının kullanılarak mimarinin desteklenmesi hedeflenmektedir.

Öte yandan tanıtılan referans mimarisi modüllerinin ve modüller arası ilişkilerin bu modüller kullanılarak geliştirilen somut ÇES yazılım mimarisi çalışmalarından elde edilen deneyimlere bağlı olarak daha güncel ve etkin hale gelebileceklerine inanılmaktadır. Bu amaç doğrultusunda SEAGENT[1] ÇES yazılımı geliştirme çerçevesi üzerinde devam eden çalışmalarımızın yeni mimari ihtiyaçlarını belirlemede ve varolan mimariyi güncellemede fayda sağlayacağı düşünülmektedir.

Kaynakça

1. Weiss, G., Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, MIT Press, USA, (1999).
2. Sycara, K., Multiagent Systems, AI Magazine, 19 (4), s. 79-92, (1998).
3. Weyns, D. ve Holvoet, T., "Multiagent Systems and Software Architecture", In proceedings of the Multiagent Systems and Software Architecture (MASSA), Special Track at Net.ObjectDays 2006, Erfurt, Almanya, s. 7-30, (2006).
4. Bass, L., Clements, P. ve Kazman, R., Software Architecture in Practice, Addison Wesley Publishing Comp., (2003).

[1] SEAGENT, Anlamsal Web ortamında çalışacak yazılım etmen sistemlerinin geliştirilmesini sağlayan açık kaynak kodlu bir Java çerçevesidir. Güncel sürümüne ve ilgili dokümantasyona http://seagent.ege.edu.tr adresinden erişilebilinir.

5. Shehory, O., Architectural Properties of MultiAgent Systems, Teknik Rapor, CMU-RI-TR-98-28, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, (1998).

6. Kolp, M., Giorgini, P. ve Mylopoulos, J., A Goal-Based Organizational Perspective on Multi-agent Architectures, Lecture Notes in Computer Science, 2333, s. 128-140, (2002).

7. Weyns, D. ve Holvoet, T., A Reference Architecture for Situated Multiagent Systems, Lecture Notes in Computer Science, 4389, s. 1-40, (2007).

8. Garcia, A., Kulesza, U. ve Lucena, C., Aspectizing Multi-Agent Systems: From Architecture to Implementation, Lecture Notes in Computer Science, 3390, s. 121-143, (2005).

9. Kendall, E. A., Malkoun, M. T. ve Jiang, C. H., Multiagent System Design Based on Object Oriented Patterns, Journal of Object Oriented Programming, 10 (3), s. 41-47, (1997).

10. Gracanin, D., Singh, H. L., Bohner, S. A. ve Hinchey, M. G., Model-Driven Architecture for Agent-Based Systems, Lecture Notes in Artificial Intelligence, 3228, s. 249-261, (2005).

11. Berners-Lee, T., Hendler, J. ve Lassila, O., The Semantic Web, Scientific American, 284 (5), s. 34–43, (2001).

12. Reed, P., Reference Architecture: The Best of Best Practices, The Rational Edge, http://www-128.ibm.com/developerworks/rational/library/2774.html, (2002).

13. Kardas, G., Goknil, A., Dikenelli, O. ve Topaloglu, N. Y., "Metamodeling of Semantic Web Enabled Multiagent Systems", In proceedings of the Multiagent Systems and Software Architecture (MASSA), Special Track at Net.ObjectDays 2006, Erfurt, Almanya, s. 79-86, (2006).

14. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. ve Stafford, J., Documenting Software Architectures: Views and Beyond, Pearson Education, USA, (2003).

15. FIPA, FIPA Abstract Architecture Specification, http://www.fipa.org/specs/fipa00001/SC00001L.pdf, (2002)

16. Kardas, G., Gümüs, Ö. ve Dikenelli, O., Applying Semantic Capability Matching into Directory Service Structures of Multi Agent Systems, Lecture Notes in Computer Science, 3733, s. 452-461, (2005).

17. Sycara, K., Paolucci, M., Ankolekar, A. ve Srinivasan, N., Automated discovery, interaction and composition of Semantic Web Services, Journal of Web Semantics, 1, s. 27-46, (2003).

18. OWL-S Coalition, OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.1/overview/, (2004).

19. WSMO Working Group, Web Service Modeling Ontology, http://www.wsmo.org/index.html, (2005).

20. Paolucci, M., Kawamura, T., Payne, T., R. ve Sycara, K., Semantic Matching of Web Services Capabilities, Lecture Notes in Computer Science, 2342, s. 333-347, (2002).

21. Gümüs, Ö., Gürcan, Ö., Kardas, G., Ekinci, E. E. ve Dikenelli, O., Engineering an MAS Platform for Semantic Service Integration based on the SWSA, Lecture Notes in Computer Science, 4805, s. 85-94, (2007)

22. Williamson, M., Decker, K. ve Sycara, K., "Unified Information and Control Flow in Hierarchical Task Networks", In Proceedings of the AAAI-96 Workshop, California, USA, s. 142-150, (1996).

23. Dickinson, I. ve Wooldridge, M., "Agents are not (just) web services: considering BDI agents and web services", In proceedings of the workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE 2005), Utrecht, the Netherlands, (2005).

24. Dikenelli, O., Erdur, R. C., Kardas, G., Gümüs, O., Seylan, I., Gurcan, O., Tiryaki, A. M. ve Ekinci, E. E., Developing Multi Agent Systems on Semantic Web Environment using SEAGENT Platform, Lecture Notes in Artificial Intelligence, 3963, s. 1-13, (2006).

25. Brussel, H. V., Wyns, J., Valckenaers, P., Bongaerts, L. ve Peeters, P., Reference Architecture for Holonic Manufacturing Systems: PROSA, Journal of Manufactoring Systems, 37, s. 255–274, (1998).

26. Amor, M., Fuentes, L. ve Vallecillo, A., Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA, Lecture Notes in Computer Science, 3382, s. 93-108, (2005).

27. Hahn, C., Madrigal-Mora, C., Fischer, K., Elvesæter, B., Berre, AJ. ve Zinnikus, I., Meta-models, Models, and Model Transformations: Towards Interoperable Agents, Lecture Notes in Artificial Intelligence, 4196, s. 123–134, (2006).

Hiyerarşik Durum Makineleri ile Olay Güdümlü Gömülü Uygulama Yazılımı Geliştirilmesi

Gökhan Bölük[1], Bülent Özümüt[2], Çağatay Çatal[3]

[1,2,3] TÜBİTAK, Marmara Araştırma Merkezi, Bilişim Teknolojileri Enstitüsü
41470, Gebze, Kocaeli
[1] gokhan.boluk@bte.mam.gov.tr, [2] bulent.ozumut@bte.mam.gov.tr ,
[3] cagatay.catal@bte.mam.gov.tr

Özet. Bu çalışmada; TÜBİTAK Marmara Araştırma Merkezi Bilişim Teknolojileri Enstitüsünde yürütülmekte bulunan bir proje kapsamında geliştirilen gömülü sistemin, çalışma senaryolarının hiyerarşik durum makineleri ile modellenmesi ve geliştirilmesi anlatılmaktadır. Proje boyunca yazılım geliştirme faaliyetlerinin donanım geliştirme faaliyetlerine paralel olarak gerçekleştirilebilmesi hedeflenmiştir. Hedef platform geliştirilene kadar uygulama mantığının modellenmesi, gerçeklenmesi ve test edilmesi sağlanmıştır. Bu amaçla, model güdümlü yazılım geliştirme yaklaşımı içerisinde yer alan platformdan bağımsız ve platforma bağımlı modellerin ayrı ayrı tasarlanması prensibi işletilmiştir.
Projenin ihtiyaçları doğrultusunda 8 veya 16 bit bir mikrodenetleyicinin kullanılmasına karar verilmiştir. Otomatik kod üreten model güdümlü yazılım geliştirme araçlarının karmaşıklığı ve ürettiği kodların büyüklüğü nedeni ile bu tip küçük gömülü sistemler için daha etkin kodlama imkanı sunan açık kaynaklı Quantum Platformu kullanılmıştır. Projenin modellenmesi aşamasında UML diyagramlarının oluşturulması ve doğrulanması Rhapsody aracı ile gerçekleştirilmiştir. Hazırlanan UML modelleri, Quantum Platformu içinde yer alan şablonlar kullanılarak kısa bir sürede doğrudan kodlanmış ve değişiklikler kolayca yazılıma aktarılabilmiştir.

1 Giriş

Yazılım sistemlerinin büyük çoğunluğu belirli olaylara tepki vererek uygun program kodlarının çalıştırılmasını sağlar. Bu tür reaktif sistemleri programlarken; çoğu zaman grafiksel bileşenler, sürükle-bırak yöntemi ile bir paletten seçilir ve grafiksel kullanıcı arayüzündeki düğmelere basılması ile birlikte bazı kod parçalarının yürütülmesi sağlanır. İçinde bulunulan bağlama (context) göre çok sayıda if-else bloklarının bu tür kodlara eklenmesi neticesinde geliştiriciler, ilgili olaylara karşı çağrılacak kod bloklarının kontrolünü kaybedebilmekte ve spagetti türünde kodlar karşımıza çıkabilmektedir [1]. Bu tür programlama paradigması, geleneksel olay-eylem (event-action) paradigması olarak adlandırılır.

Durum makineleri (state machines) ile sistemin içinde bulunduğu bağlam dikkate alınarak olaylara yanıt oluşturulabilmekte ve durum kavramı, ilişkili bağlamı etkin şekilde saklayabilmektedir. Durum diyagramları sayesinde, durum makineleri grafiksel olarak temsil edilebilmektedir. Bu diyagramlarda, düğümler durumları temsil ederken

bağlantılar geçişleri gösterir. UML (Unified Modeling Language) modelleme dili, durum diyagramları için uygun ve etkin bir gösterim şekli sunmaktadır.

Geleneksel sonlu durum makineleri (finite state machines) yönetilmesi güç hale gelmeye eğilimlidir. Birçok uygulamada var olan durumların esasında ortak özellikler içerdiği bilinmektedir. Ancak geleneksel sonlu durum makineleri bu tür benzerlikleri modelleyemez ve farklı durumlarda aynı davranışın tekrar edilmesine neden olur [1]. Belirlenen ortak özelliklerin model içerisinde tekrarlanmadan kullanılabilmesini sağlamak amacıyla hiyerarşik durum makineleri kullanılmakta ve büyük kolaylıklar sağlamaktadır. Bu çalışmada, ağırlıklı olarak hiyerarşik durum makineleri açıklanmaktadır. Bu bileşenlerin kolaylıkla gerçeklenmesi için açık kaynaklı Quantum Platform çerçevesi kullanılmıştır. Quantum Platform çerçevesi, otomatik kod üreten model güdümlü yazılım geliştirme araçlarının karmaşıklığı ve ürettiği kodların büyüklüğü nedeni ile küçük gömülü sistemler için daha etkin kodlama imkanı sunmaktadır.
Bu çalışmada; geliştirilen gömülü sistemin, çalışma senaryolarının hiyerarşik durum makineleri ile modellenmesi ve geliştirilmesi açıklanmaktadır. Gerçekleştireceğimiz sistem tasarımında dikkate aldığımız bazı önemli kriterler aşağıda sunulmaktadır:

- Proje süresince donanım ve yazılım geliştirme faaliyetlerinin birbirine paralel olarak yürütülmesinin gerekliliği,
- Proje ihtiyaçları incelendiğinde 8 veya 16 bit, düşük güç tüketimli bir mikrodenetleyicinin kullanılma zorunluluğu,
- Uygulamanın tekrar kullanılabilir, taşınabilir, basit, açık, bakım yapılabilir, güvenilir ve doğru çalışma gerekliliği.

Bir sonraki bölümde, hiyerarşik durum makineleri açıklanmaktadır. Bölüm 3'te Quantum Platformunun özellikleri ve tercih edilme sebepleri ortaya konulmaktadır. Bölüm 4'te, gerçekleştirilen uygulamadan bazı örnekler sunulmaktadır. Sonuçlar ve gelecek çalışmalar, bölüm 5'de verilmektedir.

2 Hiyerarşik Durum Makineleri

Geleneksel sonlu durum makinelerinde, benzerlikler saptanmadığı için aynı davranış farklı durumlarda tekrarlanmakta ve geleneksel durum makinelerinin yönetilebilirliğini imkansız hale getirmektedir [2]. Ortak davranış belirlenerek birçok durum karşısında yeniden kullanılabilmesi, sonlu durum makinelerinde mümkün değildir. Bu amaçla, durum grafiği (statecharts) David Harel tarafından 1980'lerde ortaya konulmuştur [3]. Bu diyagramlarda, hiyerarşik iç içe durumlar tanımlanabilmektedir. Bu nedenle durum grafiklerine, aynı zamanda hiyerarşik durum makineleri de denmektedir. UML içerisindeki durum makineleri Harel durum diyagramlarının nesne tabanlı hali olarak bilinmektedir [1]. Alt durumlar üst durumlara göre sadece farklı davranışları tanımlamalıdır. Bu yaklaşım, nesneye yönelik analiz ve tasarımdaki kalıtım kavramına benzemektedir.

İlk kez hiyerarşik durum makinesi oluşturmak, belirli bir öğrenme süreci gerektirmektedir [4]. Bu noktada kullanılan çerçeve içindeki sınıfların anlaşılması ve destekleyen modüllerin belirlenmesi zaman alıcı olabilmektedir. Ancak bir veya iki

sistem gerçeklendikten sonra, sonraki projelerde çerçeveyi kolayca kullanarak hızlı bir şekilde modelleri oluşturmak mümkündür. En fazla fayda, iş gereksinimleri ve ortam faktörleri hiyerarşik durum makinesi tabanlı sistemin değişimini gerektirdiğinde elde edilmektedir [4]. ,Değişen ihtiyaçları desteklemek üzere bu tür sistemlerde kolayca değişiklik yapılabilir. Geliştirme çevrimini kısaltması, bakım yapılabilirliği arttırması, değişen iş gereksinimlerini kolayca adresleyebilmesi nedenleriyle hiyerarşik durum makineleri oldukça yararlı modelleme elemanlarıdır [4].

Quantum Çerçevesi (framework) sayesinde, hiyerarşik durum makinelerini doğrudan kodlamak mümkün olabilmektedir. Bu çerçeve, normalde C/C++ dili ile gerçeklenmiştir. Ayrıca, .NET platformu için de C# dili ile Dr. Rainer Hessmer tarafından oluşturulmuştur [5].

"Karmaşık davranışların düzenli sonlu durum makineleriyle oluşturulması zordur. Görevler genellikle artan detay seviyesinde alt görevlere ayrıştırılabilir. Soyutlama ve modülerlik sayesinde, hiyerarşik sonlu durumlu makineler varlık geliştirme hattının karmaşıklığını önemli ölçüde azaltır. Hiyerarşiler, sistemin hesaplama gücünü arttırmaz"[6]. Bu noktada, hiyerarşik durum makineleri önemli faydalar ve kolaylıklar sunar.

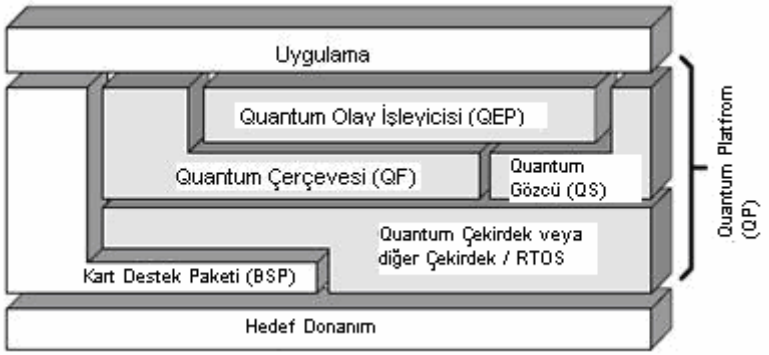
Hiyerarşik durum makineleri, giriş eylemleri (entry actions) sayesinde ilgili duruma girişte otomatik olarak bazı eylemlerin gerçekleşmesini sağlarken benzer şekilde çıkış eylemleriyle de (exit actions), ilgili durumdan çıkışta bazı eylemlerin otomatik olarak gerçekleştirilmesine olanak verir. Nesneye yönelik programlamadaki, kurucu (constructor) ve yıkıcı (destructor) fonksiyonlar, giriş ve çıkış eylemlerine benzetilebilir [1]. Giriş eylemlerinin yürütülmesi, en dış durumdan en iç duruma doğru iken çıkış eylemlerinin yürütülmesi tam ters sıradadır. Liskov yer değiştirme (substitution) prensibine göre, bir alt sınıf üst sınıfıyla yer değiştirebilmektedir. Bu prensip, iç içe geçmiş durumlara da uygulanabilmektedir.

Yeniden yapılandırma (refactoring) nesneye yönelik sistemler için geçerli olduğu gibi hiyerarşik durum makinelerine de uygulanabilmektedir [1]. Yeniden yapılandırma, ortak bir üst durum oluşturma veya özelleştirme amaçlı uygulanabilir. Tasarım kalıplarına benzer şekilde durum kalıpları da tanımlanmış olup etkin tasarımlar için kullanılabilmektedir. Hiyerarşik durum makinelerinin yeterli ölçüde geçerlenmesi, reaktif sistemleri programlamadaki düşüncelerimizde paradigma değişimini sağlayacaktır. Miro Samek [2] bu yaklaşımı Quantum Programlama olarak ifade etmektedir [1].

Hiyerarşik durum makineleri; OMT, ROOM, UML gibi nesneye yönelik yazılım geliştirme metodolojilerinin temel bileşeni haline gelmiştir. İç içe geçme (nesting) yeteneği, sadece tasarım aşamasında değil isterler (requirement) ve test fazında da kullanılmaktadır. İsterler fazında, senaryoların yapılandırılmış şekilde ifade edilmesini sağlamaktadır [7]. Hiyerarşik durum makinelerine basit bir örnek olarak, dijital saat verilebilir. Üst seviye makine 24 üst durumlu bir çevrim içerir ve her bir durum günün saatini gösterir. Her durum dakikaları gösteren 60 adet üst durum içeren bir hiyerarşik durum makinesidir. Dakikaları gösteren her durum ise saniyeleri içeren bir durum makinesidir [7].

3 Quantum Platform

Quantum Platform (QP); gerçek zamanlı uygulamalar için durum makineleri tabanlı bir alt yapı oluşturur. QP, UML durum grafikleri (statecharts) ve aktif objeleri desteklemesiyle model güdümlü geliştirme (MDD) yapabilme imkanı sağlar. Bu güçlü platform sayesinde, tasarlanan modeller pahalı Bilgisayar Destekli Yazılım Mühendisliği (Computer Aided Software Engineering-CASE) araçları kullanmadan sadece C/C++ programlama dillerini kullanarak tam ve eksiksiz olarak koda aktarılmış olmaktadır [8]. Şekil 1'de Quantum Platfromu bileşenleri ile uygulama ve donanımın ilişkisi gösterilmiştir.



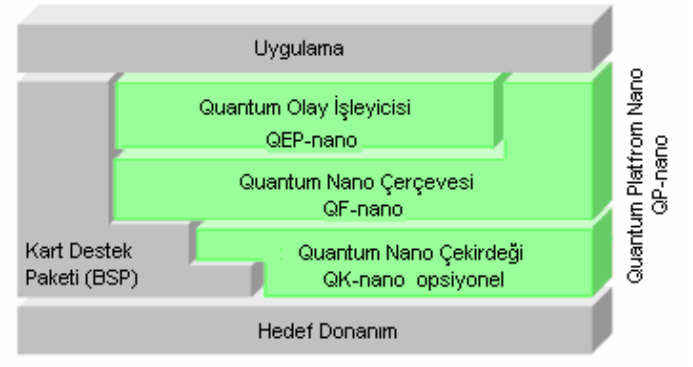
Şekil 1 Uygulama, Hedef Donanım, ve Quantum Platform Bileşenleri [9]

QP tüm bileşenleri kullanıldığında bile 4 KB lık kod ve veri alanı kapladığı için bellek açısından oldukça etkin bir kullanım imkanı vermektedir. Bu platform, farklı işlemci ve işletim sistemleri için uyarlanmış, yüzlerce uygulamada başarı ile kullanılmıştır.

Quantum Olay İşleyicisi (Quantum Event Processor-QEP); UML durum grafikleri semantiğine uygun, genel, taşınabilir ve yeniden kullanılabilir kod geliştirmeye yardımcı bir bileşendir. QEP kullanarak hiçbir tasarım aracına gerek kalmadan bellek açısından daha az yer kaplayan, daha sade ve daha anlaşılır bir kod geliştirilebilmektedir. Ayrıca QP, hiyerarşik durum makineleri oluşturma yeteneğine sahip olduğu gibi geleneksel yöntemlerle sonlu durum makineleri tasarlayabilmemizi de destekler [9].

Quantum Platformu ailesi, 8-16 bit alt seviye işlemci ve denetleyiciler ile kullanmak üzere, durum diyagramlarımızı daha kolay, daha başarılı yapılandırmamız için Quantum Platform Nanoyu (QP-nano) geliştirmiştir.

QP-nano, küçük işlemciler için uyumlu ve optimizasyonu yüksek kodlama imkanı sunmaktadır. Bu çerçevenin ana düşüncesi, durum diyagramlarını direk programlama dili seviyesinde ve C dilinde nasıl kodlayacağımızın şablonunu oluşturmaktır.



Şekil 2 Quantum Platform Nano (QPnano) Bileşenleri [10]

QPNano'ya İlişkin Önemli Özellikler

QP-nano, UML durum diyagramları tanımlarının tüm özelliklerini içermez [12]. Tasarım felsefesi olarak QP-nano en küçük boyutlu olay işleme (QEP-nano) ve en küçük boyutlu gerçek zamanlı çerçeve (QF-nano) özelliklerini destekler [10]. Şekil 2'de Quantum Platforn Nano bileşenleri ile uygulama ve donanımın ilişkisi gösterilmiştir

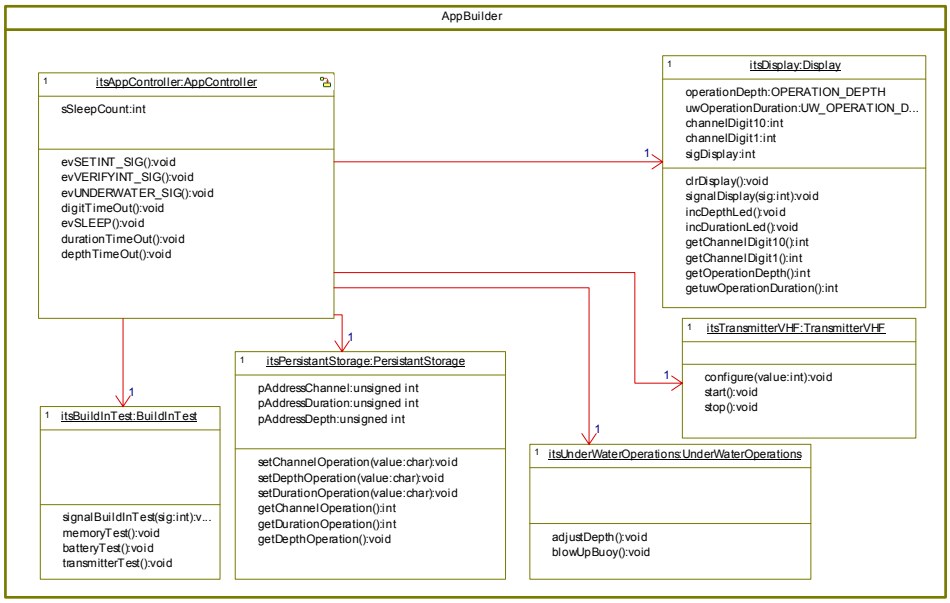
QP-nano'nun ana özellikleri aşağıda sunulmaktadır:

- Tam olarak hiyerarşik durum tanımlama desteği mevcuttur.
- UML ile tasarlanan modelin bileşenleri için sunduğu standart şablonlar sayesinde yazılım geliştirme ve bakım kolaylığı sağlar.
- Çok küçük bellek ihtiyacı gerektiren kodlar üretir. Hiyerarşik durum makinesi kodu yaklaşık 700-900 bayt, sonlu durum makinesi kodu yaklaşık 60 bayt ROM alanı işgal etmektedir.
- Durumlar ve geçişler için RAM ihtiyacı yoktur. Durum sayısı kod alanında (ROM) limitlidir.
- İsteğe bağlı olarak öncelikli çok görevli çekirdek (preemptive mutitasking kernel) QK-nano ile yapılandırılabilir. Öncelikli seçenek sadece 50-100 bayt fazladan kod belleği isterken, önceliksiz (non-preemptive) seçeneğe göre yığın için daha çok RAM alanı ister.
- QP-nano kaynak kodları, otomotiv endüstrisinde sıklıkla kullanılan MISRA (Motor Industry Software Reliability Association Guidelines for the Use of The C Language in Vehicle Based Software) standardını %98 oranında sağlar.
- Açık kaynaklı ve ticari olmak üzere ikili lisanslama mümkündür.

4 Uygulama

Bu çalışmada sistem Rhapsody aracı ile modellenmiş ve tasarlanan model Quantum Platform Nano ile kodlanmıştır. Tasarım boyunca amacımız Quantum Platformu kullanarak herhangi bir değişiklik yapmaksızın modelin koda aktarılması olmuştur. QP-nano çerçevesinin temel prensipleri, Rhapsody'nin kullandığı OXF (Object Execution Framework) çerçevesinin temel prensipleri ile çok benzediğinden anlaşılması kolay olmuştur. Aradaki fark Rhapsody'nin tasarım seviyesinde görsel modelleme dili (UML) kullanması ve otomatik kod üretmesi, diğer taraftan QP-nano'nun modellemenin direk programlama dili seviyesinde yapılmasına imkan vermesidir [8].

Gerçekleştirilen uygulama modeli, nesneye yönelik analiz ve tasarım metodolojisindeki yaklaşımlar kullanılarak tasarlanmıştır. Bu model, C programlama dili ile gerçeklenmiştir. Modeldeki sınıflar, C dili içerisindeki yapılarla (struct) ifade edilmiştir. Proje uygulama modeli, Şekil 3'de görüldüğü gibidir. Bu modelde AppController sınıfı aktif sınıf olarak tasarlanmıştır. QF aktif nesneleri türetmek için QActive temel sınıfını sunar. QActive QHSM'den türetilmiş olup, aktif nesneler hiyerarşik durum makinelerini desteklemektedir. QActive temel sınıfı ayrıca thread ve olay kuyruğu içerir.



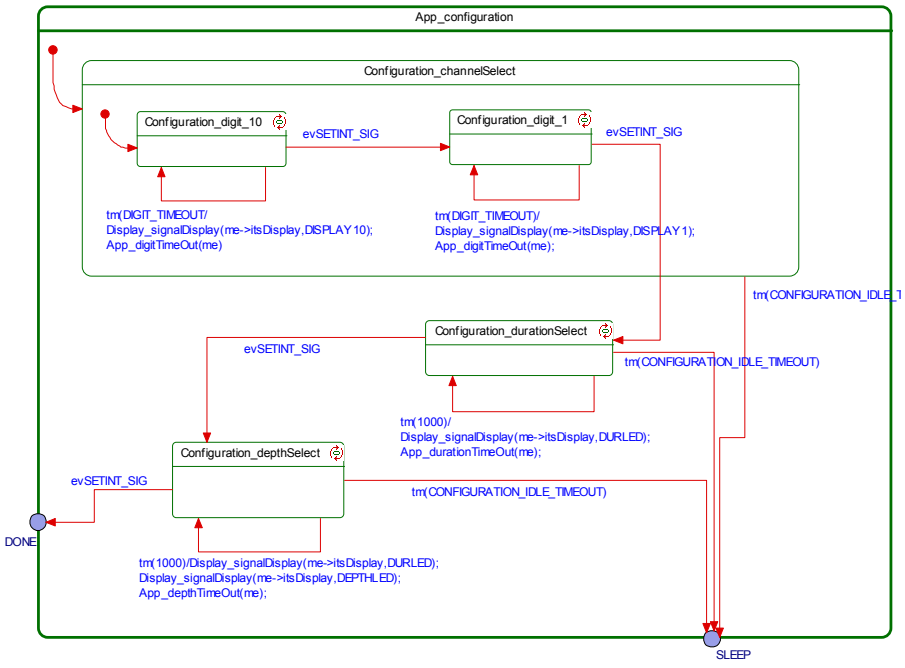
Şekil 3. Rhapsody aracı ile tasarlanan uygulama modeli

Modelde AppController sınıfı C dilinde AppController.c ve AppController.h dosyaları ile gerçeklenmiştir. AppController.h dosyası içerisinde ApplicationTag tipinde bir yapı

mevcut olup bu yapı QActive tipinde super_ isimli elemanı içermektedir. Bu sayede oluşturulan AppController sınıfının örneği (instance) aktif sınıf olarak tanımlanmış olmaktadır. Aşağıda bu bilgilere ilişkin kod bloğu verilmektedir.

```
/* AppController.h.........*/
typedef struct ApplicationTag AppController ;
struct ApplicationTag {
    QActive super_;
};
```

Sistemin çalışma zamanı, VHF çalışma kanalı gibi değerlerinin yapılandırılmasını düzenleyen App_configuration durumunun alt durum şeması Şekil 4' de gösterildiği gibi tasarlanmıştır. Bu durum uygulama modelinde yer alan hiyerarşik durumlardan sadece birisidir. App_configuration durumu içerisinde, kanal seçim için Configuration_channelSelect, süre seçimi için Configuration_durationSelect, derinlik seçimi için Configuration_depthSelect durumları tanımlanmıştır. Configuration_channelSelect durumu içerisinde Configuration_digit_10 ve Configuration_digit_1 alt durumları tasarlanmıştır.



Şekil 4. Konfigürasyon durumunun alt durum şeması

Aşağıdaki kod bloğunda durumların Quantum Platform kullanılarak nasıl tanımlanabileceği gösterilmiştir. Bu kod parçasında Configuration_channelSelect durumunun giriş olayı (Q_INIT_SIG) ile ana durumun zamanlayıcısı tetiklenmiş (QActive_arm(…)) ve alt duruma geçilmesi Q_INIT(...) sağlanmıştır. Alt Configuration_digit_10 durumunun giriş olayı (Q_ENTRY_SIG) içerisinde bir zamanlayıcı tetiklenmiş (QActive_arm(…)) ve aynı durum içerisinde tanımlanan zaman aşımı olayı (Q_TIMEOUT_SIG) ile bu tanımlanan zamanlayıcı yakalanmaya çalışılmıştır. Ayrıca kart destek paketinde (BSP) tanımlanan kesme olayı (SETINT_SIG) ile zaman aşımı olmadan bir sonraki olaya geçilmesi sağlanmıştır.

```
QSTATE  Configuration_channelSelect(AppController *me) {
  switch (Q_SIG(me)) {
          case Q_INIT_SIG: {
            QActive_arm((QActive*)me,
                        CONFIGURATION_IDLE_TIMEOUT);
            Q_INIT(&Configuration_digit_10);
            return (QSTATE)0;
        }
        case Q_TIMEOUT_SIG: { //CONFIGURATION_IDLE_TIMEOUT
            Q_TRAN(&App_idle);
            return (QSTATE)0;
        }
        }
  return (QSTATE)&App_configuration;
}
QSTATE  Configuration_digit_10(AppController *me) {
  switch (Q_SIG(me)) {
        case Q_ENTRY_SIG: {
            QActive_arm((QActive *)me, DIGIT_TIMEOUT);
            return (QSTATE)0;
        }
        case Q_TIMEOUT_SIG: { //DIGIT_TIMEOUT
             signalDisplay(DISPLAY10,&itsdisplay);
             QActive_arm((QActive *)me, DIGIT_TIMEOUT);
             return (QSTATE)0;
        }
        case SETINT_SIG: {
             Q_TRAN(&Configuration_durationSelect);
            return (QSTATE)0;
        }
        }
  return (QSTATE)&Configuration_channelSelect;
}
```

5 Sonuç ve Gelecek Çalışmalar

Bu projedeki çalışmalarımızda UML ile modelleme, model güdümlü geliştirme yaklaşımı ve durum makinaları ile tasarım üç anahtar kavram olarak ortaya çıkmıştır.

UML içerisinde, yazılım modellemesi için 13 farklı diyagram sunulmaktadır. Bu kadar çok sayıda diyagram, UML ile yeni tanışanların gözünü korkutmaktadır. Ancak bu projedeki deneyimlerimizle şunu gördük ki, iki temel diyagram ile böyle bir sistemi, hatta çok daha karmaşık sistemleri geliştirmek mümkündür. Yazılımın yapısal modeli için sınıf diyagramları, davranış modeli için ise durum makineleri adı verilen diyagramları kullanmak yeterli olmuştur. Daha karmaşık sistemlerde sistemin davranış modelini analiz etmek için ayrıca sıralama (sequence) diyagramlarının kullanımı da gerekli olabilir [11]. Durum makinelerinin etkin olarak kullanılması sıralama diyagramlarının kullanımına göre daha zordur. Durum makinelerinin tasarımında yeterli deneyiminiz yoksa öncelikli olarak sıralama diyagramlarıyla iş akışını modellemek ve sonrasında durum makinelerini tanımlamak daha uygun olacaktır. Olay güdümlü tasarım ve hiyerarşik durum makineleri kullanılarak sistemin işleyişi gerçek hayattaki gibi modellenebilmektedir. UML ile gösterilen durum diagramlarında sistemin durumları ve durumlar arasındaki geçişler kolayca görülebilir. Bundan sonra yapılacak iş, bu geçişlerde durumlara giriş ve çıkışlarda yapılacak eylemlerin tanımlanmasıdır. Bu eylemler, modelde kısa kod parçacıkları ile doldurulması gereken yordamlardan ibarettir.

Rhapsody gibi araçlarla UML ve durum makinelerinin modellenmesi aynı zamanda sistemin tasarım aşamasında test edilmesine de olanak sağlar. Bizim çalışmamızda da Rhapsody aracının animasyon özelliği kullanılarak sequence diagramları otomatik olarak oluşturulmuş ve uygulama senaryoları sınanabilmiştir. Yazılımın bakımı, doğru işleyen uygulama mantığının tasarım safhasında belirlenerek düzeltilmesiyle kolaylaşır. İleriki safhalardaki değişiklikler küçük ve kolayca anlaşılabilen kod parçacıklarıyla adreslenir. Yeni tasarımcılar ve geliştiriciler, binlerce satırlık uzun kodları incelemek yerine durum makinelerine bakarak sistemin işleyişini ve modelini kolayca anlayabilirler. Yeni isterler; yeni durumların, olayların ve eylemlerin kolayca eklenmesiyle kodun diğer kısımlarını etkilemeden gerçeklenebilir [4]. .

Model güdümlü geliştirme yaklaşımı ile platformdan bağımsız olarak oluşturulan model farklı platformlara kolayca uygulanabilmektedir. Bu uygulamada platform olarak Quantum Platform kullanılmıştır. Her ne kadar bu platforma otomatik kod üretecek bir araç olmasa da uygulamanın küçüklüğü ve modelde her bir varlığa karşılık gelen yazılım şablonlarının çok iyi şekilde ortaya konmuş olması işlerimizi kolaylaştırmıştır. Uygulama modeli bu şablonlar sayesinde, standart, güvenli, kaliteli ve tüm yazılım yaşam dönemi boyunca düşük maliyetleri sağlayacak şekilde kodlanabilmiştir.

Bu çalışmada uygulamanın karakteri gereği UML ile modelleme, hiyerarşik durum makineleri ile sistem işleyişinin modellenmesi ve Quantum platform kullanılarak modelin doğrudan koda aktarılması gerçekleştirildi. Daha karmaşık ve büyük projelerde gerçekleştirilen bu çalışmalara ilaveten durum kalıplarının kullanımını planlamaktayız. Durum kalıpları gerçek hayatta karşılaştığımız ortak problemlere karşı tekrar kullanılabilir ve paylaşılabilir durum makinesi çözümleri sunmaktadır. Nesne tabanlı

tasarım kalıpları sınıf ve nesneleri yapılandırırken, durum kalıpları durum, olay ve geçişlerin yapılandırılmasına odaklanmaktadır. Durum kalıpları, davranış modellerinin kalıtımı yoluyla durum makinelerinin tekrar kullanılabilirliğine imkan tanımaktadır. Bunun yanısıra, Rhapsody OXF (Object Execution Framework) ya da IDF (Interrupt Driven Framework) çatısını, kullanacağımız platforma uyarlayarak otomatik olarak kodlama yapmayı hedefliyoruz. Büyük çaplı projelerde kodlamanın elle yapılması, projenin ileriki safhalarında modelin kodla eş zamanlı olarak güncellenememesi riskini beraberinde getirmektedir. Otomotik kod üretimi, model ile kodun her zaman birbirine eş olmasını garanti edecektir.

Kaynakça

1. Samek, M., "Hierarchical State Machines: A Fundamentally Important Way of Design", www.quantum-leaps.com/resources/samek0405parc.pdf
2. Samek, M., "Practical Statecharts in C/C++: Quantum Programming for Embedded Systems", CMP Books, 2002.
3. Harel, D., "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Programming, 8, 1987, sf. 231-274.
4. www.automation.com/sitepages/pid1522.php
5. www.hessmer.org/dev/qhsm
6. Champandard, A. J., "AI Game Development: Synthetic Creatures with Learning and Reactive Behaviours", New Riders, Indianapolis, Indiana, 2004.
7. Alur R., Yannakakis, M., "Model Checking of Hierarchical State Machines", ACM Transactions on Programming Languages and Systems, 23(3): 273-303, 2001
8. Lan, H., "An Open Design Metodology For Automotive Electrical/Elactronic System Based On Quantum Platfrom", 2007
9. http://www.quantum-leaps.com/doc/QP_Overview.pdf
10. http://www.quantum-leaps.com/doxygen/qpn_man_4.0.00.chm
11. Dougles B. P. , "Real Time Design Patterns", Addison-Wesley, 2004
12. http://www.omg.org/docs/ptc/03-08-02.pdf

Görev Kritik ve Gömülü Sistemler İçin Hata Yönetimi Kılavuz Mimarisi: T^{5}D

Metin Tekkalmaz[1], Özgür Kaya[2], Mustafa Dursun[3], Tuçe Sarı Tekkalmaz[4], Ali Doğru[5]

[1, 3, 4] ASELSAN A.Ş., REHİS Grubu, Ankara
[2, 5] Bilgisayar Mühendisliği Bölümü, ODTÜ, Ankara
[1] tkalmaz@aselsan.com.tr, [2] okaya@ceng.metu.edu.tr, [3] mdursun@aselsan.com.tr,
[4] tsari@aselsan.com.tr, [5] dogru@ceng.metu.edu.tr

Özet. Bu makalede görev kritik ve gömülü sistemlerde hata yönetimi kabiliyetlerini oluşturacak altyapıların geliştirilebilmesi için bir yazılım ürün hattı mimarisinin tasarımı anlatılmaktadır. Farklı uygulamaların farklı hata yönetimi gerekleri olacaktır ve bu gerekleri karşılayacak hata yönetimi altyapılarının güvenilir, etkin ve çabuk bir şekilde ortaya çıkarılması hedeflenmiştir. Bu nedenle alana yönelik hata yönetimi gereksinimleri modellenip esnek bir kılavuz mimari, uyumlu temel bileşenler ile birlikte tasarlanmıştır. Alana özel sabit ve değişkenlik gösteren farklı kabiliyetler, değişik haberleşme mekanizmaları ile çalışabilecek şekilde ele alınmıştır. Geliştirilen modeller 4+1 mimari görüş açıları ile uyumludur. Bu çalışma, çakılmayan ve kendi kendine iyileşen görev kritik ve gömülü sistemlerin geliştirilebilmesi yönünde şekillenen bir çalışmadır.

1 Giriş

Yazılımlar hata içerir. Bazı küçük uygulamalar haricinde bir uygulamanın hatasız olduğunu ispatlamak imkânsızdır ve hataların da tamamen giderilemeyeceği kabul edilir. Ancak bu hatalar çalışma esnasında arızalara neden olurlar. Bazı yazılımlar için bu arızalar fazla zarara yol açmaz ve uygulamayı yeniden başlatmak gibi yöntemlerle kullanıma devam edilir. Bazı uygulamalar için ise arıza, ciddi sonuçları olabilecek kabul edilemez olaylardır. Görevin iptal edilmesi, bazen de maddi zarar ve hatta can kaybı söz konusudur. Hatalardan kurtulmanın mümkün olmadığı bir gerçek olduğu akılda tutularak hataların varlığına rağmen arıza oluşturmamaları ya da arıza oluştuğunda çalışan sistemin bundan önemli derecede etkilenmemesi yönünde tedbirler alınabilir: Yazılımlar hataya dayanıklı olarak tasarlanabilir.

Bu çalışmada hedeflenen hataya dayanıklılık, görev kritik ve gömülü sistemlerin oluşabilecek arızaları önceden tahmin ederek önlem alması, bir arıza oluşursa da sistemin kendisini düzeltmesi yönündedir. Bu tür ihtiyaçları olacak birçok projenin geliştirileceği de bilinmektedir. Bu yüzden bir ürün hattı organizasyonu ile oluşturulacak hata yönetimi kabiliyetlerinin farklı ihtiyaçlar doğrultusunda uyarlanarak kullanılabilmesi hedeflenmiştir. Bu kapsamda ürün hattı yaklaşımının gereği olarak uygulama alanı sınırları ile belirlenmiş, kabiliyetlerin sabit ve değişken noktaları ile birlikte gösterildiği yetenek modeli oluşturulmuştur. Yetenek modeli ile organik bağlar bu aşamada kurulmamış ancak bu modelden hareketle kılavuz mimari ortaya konmuştur. Farklı alanlara da uyarlanabileceği bilinmekle birlikte, hedef alan görev

kritik ve gömülü sistemler için hata yönetimi olarak belirlenmiştir. Bu doğrultuda hız, haberleşme, dağıtıklık ve özelleşmiş donanımlar ile çalışma ihtiyacı gibi konulardan kaynaklanan ek zorluk ve karmaşıklık, bahsi geçen ürün hattı oluşturulurken değişik aşamalarda dikkate alınmıştır.

Bir arıza oluştuğunda sistemin kendi kendine bu arızayı gidermesi ve bu esnada da görevin elden geldiğince zarar görmemesi istenir. Önleme girişimleri yanında yerleşmiş hata yönetimi mekanizmaları ile, örneğin önemli bir öğenin yedeklenmesi ve arıza oluşunca otomatik olarak devreye alınması sayesinde, bu istekler gerçekleştirilir. Çalışma, kullanıcı müdahalesinin en aza indirgendiği, kendi kendine gerçekleşen bu faaliyetlerin yanında doğası gereği kullanıcı müdahalesi gerektiren süreçleri de ele almıştır. Bu süreçler, normal işleyişi aksatan teşhis veya düzeltilme faaliyetleri ile ilgili olarak kullanıcı kararını gerektiren durumları kapsar. Kullanıcı girdisi sayesinde normal sistem çalışmasına etkilerine rağmen teşhis veya düzeltme faaliyetinin başlatılabileceği bilgisi elde edilir.

Bu çalışmanın ayrı bir özelliği de alışılmış hata yönetimi mekanizmaları yanında tahmine dayalı bir önleme kabiliyetini de içermesidir. Her ne kadar sistem kendi kendisini düzeltebilse de, bir arıza oluştuktan sonra düzeltme operasyonunun maliyeti bulunmaktadır. Bu maliyet, haberleşme ya da işlemci yükü oluşturmak yanında bu ek yüklerden dolayı sistemin azaltılmış kabiliyetlerle çalışması anlamına da gelebilir. Eğer yapılabilirse, sistem işleyişi gözlenerek olası arıza gelişmesi zamanında anlaşılıp önleyici davranışlarda bulunulabilir. Bu yaklaşımın hedefi ise daha az bir maliyet ile gerçekleştirilebilecek olan önleme işlemi sayesinde daha büyük maliyetli arıza düzeltme işlemlerinden elden geldiğince kurtulmaktır.

Bu çalışmada yer alan başlıca hata yönetimi etkinlikleri, Takip (Belirti İzleme), Tespit, Teşhis, Tahmin, Tamir ve Düzeltme işlemleridir. Bu işlemlerin baş harfleri ile hata yönetimi yaklaşımının ismi T5D olarak anılmaktadır.

Hedeflenen ürün hattındaki farklı uygulamaların farklı altyapı gereklerinin olduğu bilinmektedir. Birimlerin haberleşmesi için değişik ara katmanlar kullanılabilecektir. Dolayısı ile hata yönetimi kılavuz mimarisi de uygulamanın gerektireceği farklı ara katmanlar ve haberleşme mekanizmalarına uyum gösterebilmelidir. Bu nedenle tasarımda önem verilen esneklik, haberleşme konusunda da korunmuştur.

Kuruluş bünyesinde hata yönetimine bu çalışma ile başlayan yapısal yaklaşımdan önce de önem verilmiş ve bazı mekanizmalar kurulmuş idi. Bu konuyu bir ürün hattı düzeyinde ele alınca daha önce kurulmuş olan kabiliyetlerin de yeni mimari içerisinde yer alması beklentisi doğal olarak ortaya çıkmıştır. Bu mekanizmalar arasında önemli bir yer tutan Cihaz İçi Test (CİT) mekanizmaları ile birimlerin açılış esnasında ve isteğe bağlı olarak bazı hata kontrolleri yapması söz konusudur. CİT yapıları da birer hata yönetim birimi olarak değerlendirmeye katılmışlardır. Bu yaklaşım daha genel olarak ele alınmış, sonucunda da karmaşık bir sistemin hata yönetiminde farklı hata yönetim birimlerinin hiyerarşik olarak birlikte çalışabilmesi de düşünülmüştür. Bu çalışmanın mevcut araçlarla kolayca gerçekleştirilemeyecek önemli yönleri, bu türden bir bütünleşme altyapısı, tahmin ve önleme konusundaki yapısal yaklaşım, farklı altyapılarla birlikte çalışabilirlik ve bir yazılım üretim hattı esnekliğidir.

Bu çalışma kapsamında tasarlanan hata yönetimi için kılavuz mimari, Radar Elektronik Harp Fonksiyonel Referans Modeli'nin (REFoRM) [1] hata yönetimi kısmını karşılamaktadır. REFoRM, ASELSAN REHİS Grubu'nda Radar ve Elektronik Harp projeleri için ortak, yeniden kullanılabilir bileşenlerin olduğu ve yazılım ürün hattı temel alınarak geliştirilmiş bir modeldir.

Makalenin geri kalanında sunum şu şekildedir: 2. Bölüm ortaya konulan hata yönetimi kılavuz mimarisini değişik mimari bakış açıları kullanarak açıklamaktadır. 3. Bölüm çalışmayı, bundan sonra hedeflenen çalışmalarla birlikte değerlendirmektedir. Son bölüm ise bu çalışmaya önemli katkılarda bulunanlara teşekkürleri içermektedir.

2 Kılavuz Mimari

Hâlihazırda değişik hataya dayanıklılık teknikleri [2], [3] ve değişik hata yönetimi araçları [4], [5] bulunmaktadır ve bunlar genel olarak arızayı tespit etmek, oluşan hasarı onarmak veya hasarın yayılmasını önlemek üzere tasarlanmıştır.

Bu çalışma kapsamında hazırlanan hata yönetim kılavuz mimarisinin çıkış noktası radar ve elektronik harp sistemleri gibi gömülü ve görev kritik sistemlerde oluşabilecek bir arızanın, işleyişi olumsuz yönde etkilemesi sonucunda sistemin işlevini yerine getirememesini engellemektir. Bu kapsamda temelde yazılım ancak gerektiğinde donanım arızalarını algılayabilecek, bunları düzeltebilecek ve bu şekilde kendi kendini iyileştirebilecek sistemler üzerinde çalışma başlatılmıştır.

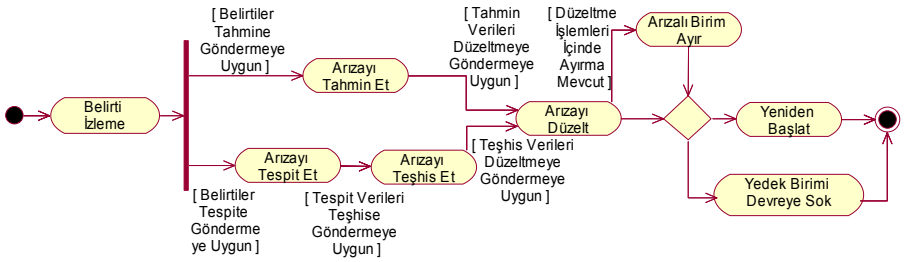
Hata yönetimi kılavuz mimarisi ürün hattı yaklaşımına uygun bir şekilde geliştirilmiştir [6]. Radar ve elektronik harp projelerinde hata yöntemi konusunda kapsamlı bir alan analizi yapılmıştır. Alanda bulunan benzerlikler ve değişebilir noktalar belirlenmiştir. Bunların hepsi bir kılavuz mimari çerçevesinde toplanmıştır.

Hata yönetimi içindeki hangi bileşenlerin uygulanacağının ve uygulanma sırasında hangi yöntemlerin kullanılacağının seçilebilir bir şekilde olması düşünülmüştür. Bu şekilde birçok farklı kapsamlı projede uygulanması düşünülen hata yönetiminin istenilen yeteneklerinin alınıp, diğerlerinin dışarıda bırakılması ile yine aynı kılavuz mimari ışığında hata yönetimi işlevi gerçekleştiriminin sağlanması amaçlanmıştır. Ayrıca projeler farklı alt yapılar kullanabileceğinden, hata yönetimi kılavuz mimarisinin uygulama esnasında uygulamalarla bütünleşik ya da uygulamalardan ayrı, dağıtık ya da merkezi, doğrudan metot çağrımı ya da soket haberleşmesi tarzında seçenekleri desteklemesi hedeflenmiştir.

Bu makalede mimari görüş açılarına uygun değişik görüşleri açıklayıcı UML tabanlı 4+1 modeli kullanılmıştır. Bu model kapsamında kullanım durumu (use case) merkezli olmak üzere tasarım (design), gerçekleme (implementation) ve konuşlandırma (deployment) açıları verilmiştir. Hata yönetiminin süreçsel (process) açısından bu makale kapsamında bahsedilmemiştir.

2.1 Kullanım Durumları

Hata yönetiminin genel bir akışını anlatmak adına Şekil 1'de bir etkinlik çizgesi gösterimi yer almaktadır.



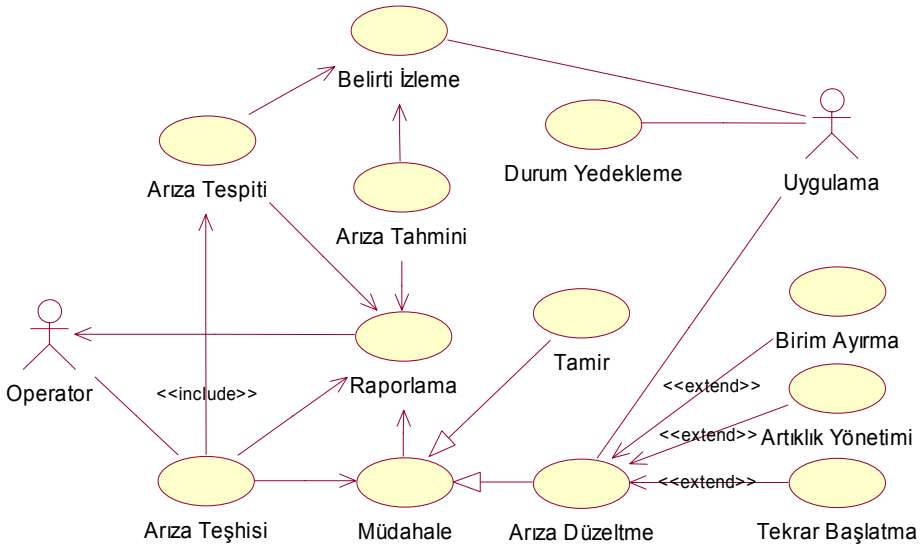
Şekil 1. Hata Yönetimi Etkinlik Çizgesi

Belirti izleme durumunda, sistemdeki arızaya neden olabilecek belirtiler izlenir. Bu belirtiler işletim sistemindeki görevlerin durumlarından, bellek kullanımına, donanımların sıcaklık verilerinden, haberleşmelerdeki veri doğruluğuna kadar değişik şekillerde olabilir. Kullanılacağı sistemin ihtiyaçlarını karşılayacak şekilde tanımlanan kurallara uygun olarak izlenen belirtiler; tahmine, tespite ya da her ikisine de gönderir. Tahmin durumunda, tahmin içinde bulunan kurallar ile bir arızanın oluşmak üzere olduğunun tahmini yapılır. Tespit durumunda, belirtiler sonucunda bir arızanın varlığı belirlenir ve bu bilgi teşhise gönderilir. Teşhis işleminde, bulunan arızanın asıl kaynağı, yani nedeni aranır. Bu arama işlemi kurallar, algoritmalar ve geçmiş arıza kararları kullanılarak yapılır. Teşhisten gelen ve teşhis edilen arızanın kaynağı ya da tahminden gelen tahmin edilen olası arıza kaynağı düzeltme işlemine tabi tutulur. Düzeltme durumunda, arıza sistemin bütününe ya da bir kısmına zarar veriyor ise arızalı birim ayrılabilir, arızalı birimin yedeği var ise bu devreye sokulabilir ya da arızalı birim yeniden başlatılabilir. Yeniden başlatma işlemi arızalı birimin durumunun yedeklenip yedeklenmediğine ya da yedeklemenin çeşidine göre sıcak, ılık ve soğuk olarak yapılabilir.

Düzeltme, yaptığı düzeltmenin başarılı olup olmadığını teşhisten gelen bilgi aracılığı ile anlayacaktır. Düzeltme, bir düzeltme faaliyetinden sonra arıza kalkmadıysa önceki düzeltme faaliyetini tekrarlayabilir ya da daha kapsamlı düzeltme faaliyeti gerçekleştirebilir.

Bunların dışında çevrim dışı olarak gerçekleşen tamir işlemi de yer almaktadır. Tamir, arızaya neden olan hatanın giderilmesi işidir. Yazılım için hata ayıklama, yeniden derleme veya donanım için arızalı parçanın değiştirilmesi gibi işlemleri içermektedir. Yine etkinlik çizgesinde yer almayan ancak her durumda gerçekleşebilecek Raporlama, gerek hata yönetimi içerisinde sonradan kullanılmak üzere, gerekse kullanıcıya yönelik bilgilerin saklanması işi ile ilgilenir.

Şekil 2'de hata yönetimi kapsamında kullanım durumları yer almaktadır. Bu kulanım durumları hata yönetiminin ana kabiliyetleri olarak görülebilir.



Şekil 2. Kullanım Durumu Çizgesi

Yukarıda anlatılan temel işlemlerin hepsi bir yetenek olarak hata yönetimi mimarisi içinde yer almaktadır. Projeler istedikleri yetenekleri alarak kullanabilirler. Örneğin hata tahmin özelliğini bir proje kullanırken bir diğeri kullanmayabilir. Benzer şekilde, yedek birimi olmayan projeler yedekleme ile ilgili mimari kısımları projelerine dâhil etmeyebilirler.

2.2 Gerçekleme Açısı

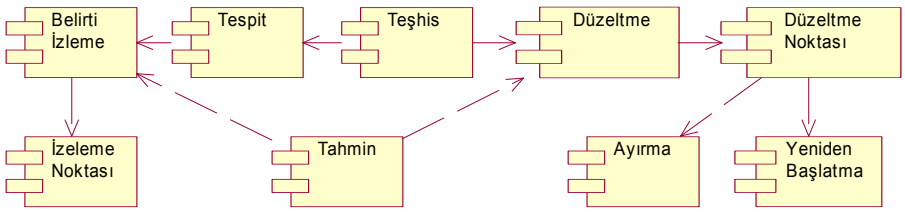
Geliştirilecek uygulamalara uygun bir hata yönetimi ortamı sağlamak üzere temelde Hata Yönetimi bileşenleri, Uygulama Yazılımı, İşletim Sisteminden oluşan üç unsurun birlikte çalışma durumunda olacağı göz önünde tutulmalıdır. Hata Yönetimi bileşenleri şunlardır: Teşhis, Tespit, Tahmin, Düzeltme, İzleme. Bu temel bileşenler haricinde Düzeltme bileşeni aracılığı ile kullanılan Ayırma ve Yeniden Başlatma bileşenleri ile İzleme bileşeni aracılığı ile kullanılan İzleme Noktası bileşeni bulunmaktadır.

Tespit, Teşhis ve Tahmin bileşenleri, Hata Yönetimi'nin arıza ile ilgili kararları verme yeteneğine sahip bileşenleridir. Teşhis bileşeni, tespit edilen arıza ile ilgili olarak kural tabanlı bir algoritma çalıştırır ve gerekiyorsa daha detaylı tespit isteklerinde bulunabilir, dolayısı ile farklı izlemelere de neden olabilir. Teşhis bileşeni kullanıcı ile etkileşimin esas noktasıdır.

Düzeltme, Ayırma ve Yeniden Başlatma bileşenleri alınan arıza ve arıza ihtimali kararları dâhilinde gerekli düzeltmelerin sağlıklı bir biçimde gerçekleştirir. Düzeltme bileşeni hangi düzeltmelerin yapılacağına kural tabanlı bir biçimde karar verdikten sonra arızanın diğer birimlere yayılmaması için Ayırma bileşenini kullanır. Yeniden

Başlatma bileşeni ise arıza oluşmuş birimlerin yeniden başlatılarak ilk durumlarına dönmelerini sağlar.

Hata Yönetimi bileşenlerinin birbirleri ile olan ilişkileri Şekil 3'te gösterilmektedir.



Şekil 3. Hata Yönetimi Yazılımı Bileşenleri ve Bileşen İlişkileri

Hata Yönetimi bileşenleri, Uygulama Yazılımı ve İşletim Sisteminin birbirleri ile etkileşimleri mantıksal olarak katmanlı bir yapı içinde ifade edilebilir. Bu yapıyı oluşturan katmanlar Şekil 4'te gösterildiği gibi Yönetim Katmanı, Temas Katmanı ve Uygulama Katmanıdır.

Şekil 4. Hata Yönetimi Katmanları

Uygulama Katmanında Uygulama Yazılımı ve İşletim Sistemi yer almaktadır. Temas Katmanı, Hata Yönetimi bileşenlerinden İzleme ve Düzeltme bileşenlerini içerir. Yönetim Katmanı ise Hata Yönetimi bileşenlerinden Tespit, Teşhis ve Tahmin bileşenlerini içerir. Yönetim Katmanında yer alan bileşenler yönetsel işlevleri yerine getirmektedir. Yönetsel işlevler için gerekli bilgilerin alınmasını ve yönetsel işlevler sonucunda yapılması gereken müdahaleleri Temas Katmanında yer alan bileşenleri kullanarak gerçekleştirmektedirler. Bu sebeple Temas Katmanında yer alan bileşenlerin, Uygulama Yazılımları ve İşletim Sistemi ile daha sıkı bir ilişki içinde olması söz konusudur.

Katmanların kendi içinde ve aralarındaki etkileşim, Uygulama Yazılımları, İşletim Sistemi ve Hata Yönetimi bileşenleri arasında haberleşme ihtiyacı doğurmaktadır. Haberleşme ihtiyacı, Hata Yönetimi bileşenlerinin konuşlandırmasına, kullanılacak

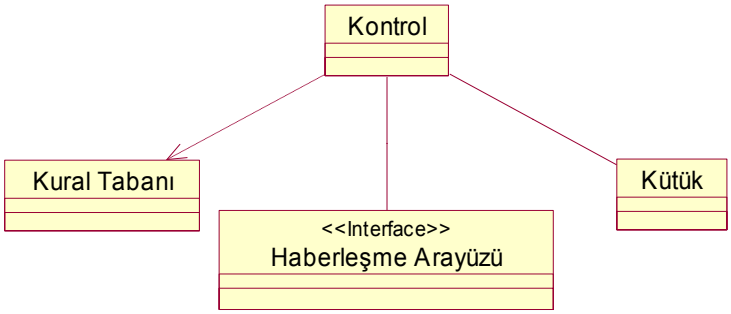
fiziksel arayüzlerin çeşidine (Ethernet, PCI Express gibi) ve kullanılacak ara katman mimari modellerine göre (Servis Odaklı, Mesaj Odaklı, Veri Odaklı gibi) şekillendirilebilir olmalıdır. Mesaj Ara Katmanı, bu ihtiyaçlara cevap veren ve katmanları birbirleri ile bağlayan önemli bir bileşendir.

2.3 Tasarım Açısı

Kılavuz mimaride kullanılan tasarım açısı, mimarinin kavramsal birimleri ve bu birimler arasındaki etkileşim ve ilişkileri yansıtmaktadır ve sınıf çizgeleri aracılığı ile modellenmiştir. Sınıf çizgelerinde yer alan sınıflar, ayrıca gerçekleştirme açısı modelinde de değişik bileşenlerin içyapılarını tanımlamak üzere kullanılmışlardır. Sınıflar, mimari yapının temel ögelerini oluşturmaktadırlar. Bu sınıflar, kavramları gerçekleştirmek için gerekli olan veri merkezli yapısal birimlere karşı düşmektedirler. Sınıf çizgeleri bu makale kapsamında verilmemiş ancak aşağıda üç ana grup olarak listelenmiştir:

- Yönetim ile ilgili sınıflar
    - Tahmin, ‘Tahmin Kuralları’ ve ‘Tahmin Kütüğü’
    - Tespit, ‘Tespit Kuralları’ ve ‘Tespit Kütüğü’
    - Teşhis, ‘Teşhis Kuralları’ ve ‘Teşhis Kütüğü’
- Müdahale ile ilgili sınıflar
    - ‘Artıklık Yönetimi’, ‘Artıklık Kuralları’
    - Ayırma, ‘Ayırma Kuralları’, ‘Ayırma Kütüğü’
    - ‘Tekrar Başlat’, ‘Tekrar Başlat Kuralları’ ve ‘Tahmin Kütüğü’
    - 'Düzelt, ‘Düzeltme Kuralları’ ve ‘Düzeltme Kütüğü’
    - ‘Durum Yükleyici’
- İzleme ile ilgili sınıflar
    - ‘Belirti İzleme’, ‘İzleme Noktası’ ve ‘Belirti Kütüğü’

Yukarıda belirtilen sınıf gruplandırmalarından da görüleceği gibi bu yapı, tasarım kalıpları ile yapılandırmaya uygunluk göstermektedir. Kılavuz mimaride tutarlı bir şekilde ortaya çıkan örüntü (bileşen, bileşen kütüğü ve kuralları), tanımlanan bileşenlerin Şekil 5’te gösterildiği gibi bir ‘Kural Tabanlı Kontrol Kalıbı’ şeklinde algılanabileceğini ortaya koymuştur.

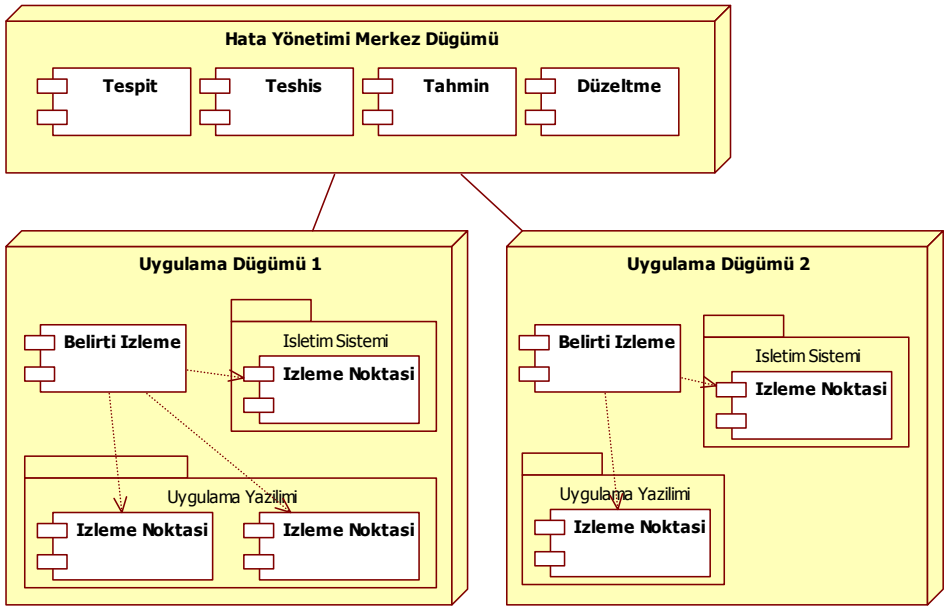


Şekil 5. Kural Tabanlı Kontrol Kalıbı

İşleyişin değişik ayrıntıları düşünüldüğünde literatürde tanımlı olan yaygın kalıpların da kullanılabileceği anlaşılmaktadır. Örneğin, durum yedeklemesi için 'memento' ve 'state' tasarım kalıplarına [7] uygun bir geliştirme yapılabilir. Haberleşme ihtiyaçları doğrultusuna 'proxy' kalıbı da değerlendirilebilir. Farklı platformlar kullanmak durumunda kalacak dağıtık bileşenlerin haberleşme noktalarında ise arayüz uyumu sağlayacak olan 'adapter' kalıbı da değerlendirilebilecek katkılar sağlayacaktır.

2.4 Konuşlandırma Açısı

Hata Yönetimi, dağıtık sistemlerde kullanılabilir bir yapıya sahiptir. Bu yapı Hata Yönetimi bileşenlerinin de dağıtılmasını ve değişik konuşlandırma ihtiyaçlarını gerektirmektedir. İzleme ve düzeltme işlemlerini gerçekleştiren Belirti İzleme, İzleme Noktası, Düzeltme ve Yeniden Başlatma bileşenleri işletim sistemi ve uygulama yazılımı ile doğrudan çalışacağından bu bileşenlerin çoklanarak her düğümde konuşlandırılması gerekmektedir. Yönetim Katmanında bulunan Tespit, Teşhis ve Tahmin bileşenleri ise merkezi bir düğümde yer alarak yönetsel faaliyetleri daha etkin bir biçimde gerçekleştirebileceklerdir. Ayrıca Merkezi düğümde dağıtık Düzeltme bileşenlerinin faaliyetlerini kontrol edecek bir düzeltme işlemine de gerek vardır. Şekil 6'da Belirti İzleme ve İzleme Noktası bileşenlerinin çoklanarak düğümlere dağıtıldığı ve Tespit, Teşhis, Tahmin, Düzeltme bileşenlerinin merkezi bir düğümde yer aldığı bir konuşlandırma örneği yer almaktadır.



Şekil 6. Hata Yönetimi Bileşenleri Konuşlandırması

3 Sonuç

Gereksinimler ve alan modelleme çalışmaları ile başlayan ürün hattı tanımlamasını gerekli kılavuz mimari ve temel bileşenlerin tasarımı izlemiş ve esnek bir hata yönetimi altyapısı tasarımı oluşturulmuştur. Farklı ihtiyaçlara yanıt verebilme prensibi çalışma boyunca akılda tutulmuş ve bunun sonucunda da esnek bir yapı sağlayan yapıtaşları göz önünde bulundurulmuştur. Örnek olarak, bu yapı taşları arasında bulunan haberleşme, farklı ara katmanlar ile çalışabilmenin yanında doğrudan işlev çağırma yöntemini de kullanabilecek, değişik haberleşme protokolleri ile uyumlu olabilecektir. Hiyerarşik bir hata yönetimleri tümleştirilmesi de düşünülmüş, bazı kabiliyetlerin dağıtık bir yapıda da yerleştirilebileceği hesaba katılmıştır. Kural tabanlı bir yaklaşımla esneklik temin edilmiş ve bunun için de bir tasarım kalıbı tanımlanmıştır. Ana bileşenlerin tümünde bu kural tabanlı tasarım kalıbı uygulanabilir olarak gözlemlenmiştir.

Çalışma gerçekleme aşamasına girmiştir. Yapılan değerlendirmeler ve önerilen bileşenlerin geliştirilmesine devam eden bir projedeki örnek uygulamalarından elde edilen neticeler doğrultusunda tasarlanmış bulunan kılavuz mimarinin ve destekleyici bileşenlerin, farklı hata yönetimi ihtiyaçlarına yanıt verebileceği kanısı oluşmuştur. Ayrıca, hata yönetimi kılavuz mimarisini temel alan ürünlerin ASELSAN içerisinde eşzamanlı devam eden bir çalışma sonucunda ortaya konulacak kaynak yönetimi yazılımının temel bir parçası olacağı da görülmüştür.

Çalışmanın devam edecek olan aşamalarında ürün hattının temel yapıları gerçekleştirilecek ve farklılaşma yönetiminin de yapılabileceği bir yetenek modeli üzerinden istenen hata yönetimi altyapısının çabuk konuşlandırılmasına yönelik bir geliştirme faaliyetine başlanacaktır.

Teşekkürler

Bu çalışmanın dâhil olduğu HAYAT (HAta Yönetimi AlTyapısı) projesinin hayata geçirilmesini sağlayan ASELSAN'a ve hata yönetimi kılavuz mimarisinin ortaya konması aşamasında değerli katkılarını esirgemeyen Reyhan Ergün, Onur Aktuğ, H. Özgür Gören ve Levent Alkışlar'a teşekkür ederiz.

Kaynakça

1. Alkışlar L., "ASELSAN'da Yazılım Mimarileri Uygulamaları", Ulusal Yazılım Mimarileri Konferansı, İstanbul, 2006.
2. Kalinsky, D., "Principles of High Availability Embedded Systems Design", Embedded World Conference, Nürnberg 2006.
3. HA Forum, "Providing Open Architecture High Availability Solutions", Revizyon 1.0, Şubat, 2001.
4. Maia R., Henriques L., Costa D., ve Madeira H., "Xception? - Enhanced Automated Fault-Injection Environment", 2002 International Conference on Dependable Systems and Networks (DSN 2002), Bethesda, MD, USA, 23-26 June 2002.
5. GoAhead Software Incorporated, SelfReliant Technical Product Description, Washington, September 2005.
6. Pohl K., Böckle G., ve v. d. Linden F., "Software Product Line Engineering: Foundations, Principles, and Techniques", Springer, Berlin Heidelberg New York, 2005.
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, Reading, Massachussetts, 1995.

Bir "Mimari Evrim" Hikayesi

Ümit Demir[1], Onur Aktuğ[2]

[1,2] ASELSAN A.Ş., REHİS-GYM, 06172, Macunköy, Ankara
[1] udemir@aselsan.com.tr, [2] oaktug@aselsan.com.tr

Özet. Bu makalede, ASELSAN'da geliştirilen uygulama yazılımlarında kullanılan mimarilerin gelişimi anlatılmaktadır. Tüm uygulama alanını kapsayan büyük yazılımlardan, zamanla alan bilgisinin artması, alanın daha detaylı, küçük parçalara bölünmesi ve yazılım ihtiyaçlarının da buna paralel bir gereksinim içine girmesiyle birlikte bileşen tabanlı yazılımlara geçiş ihtiyacı belirmiştir. Benzer şekilde bileşenler, farklı projelerin farklı ihtiyaçlarını karşılama gereğiyle çerçeve ve eklentilerden oluşan eklenti tabanlı yazılım bileşenleri haline dönüşmüştür. Son olarak da servis tabanlı mimarilere geçilerek birbirleri ile etkileşen çok sayıda yazılım bileşeninden oluşan bir yapıya geçilmiştir. Bu gelişme ve değişmenin en önemli tetikleyicileri yazılım karmaşıklığını azaltma, tekrar kullanılabilirliği arttırma, daha kalite yazılımları daha kısa sürede hazırlama, farklılaşan ihtiyaçları karşılama gibi kalite faktörleri ile alan analizi olmuştur.

1 Giriş

Yazılım mimarileri zaman içinde değişmekte ve gelişmektedir. Tüm yetenekleri içinde barındıran devasa yazılımları hedefleyen dikey yaklaşımlar zamanla terkedilmiş, benzer ve birbirleri ile ilgili, gruplu yetenekleri barındıran yatay yaklaşımlar ortaya çıkmış ve katmanlı mimariler kullanılmaya başlanmıştır [1]. Günümüzde ise çok sayıda yazılım biriminin birbirleriyle etkileştiği yapılar olan servis tabanlı mimariler yaygın olarak kullanılmaya başlanmıştır. Mimarilerin bu gelişimlerinde özellikle tekrar kullanım, karmaşıklığı azaltma, daha kaliteli yazılımları daha hızlı geliştirme kaygıları ön planda olmuştur. Ürün gerekleri, kalite faktörleri ve alan analizi çalışmaları bu tür bir mimariye yönelten etmenler olmuştur. Bu makalede ASELSAN'da geliştirilen "Görev Yazılımlarında" kullanılan mimari yaklaşımların zaman içindeki evrimleri anlatılmaktadır.

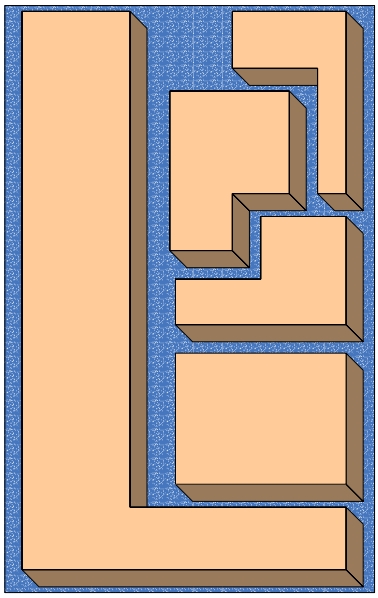
2 ASELSAN Görev Yazılımları Mimarilerinin Gelişim Evreleri

Projelerde tek ve her yeteneği içinde barındıran yazılımlar zaman içinde tekrar kullanılamamış ve idamelerinde zorluklar yaşanmıştır. Proje teslimatları gerçekleştirilmiş ancak yeni projelerde, var olan yazılımların tekrar kullanılabilirliği sınırlı kalmış, yazılım hatalarını bulmak amacıyla yapılan testler uzun süreler almıştır.

Zamanla nesne yönelimli tasarımlar kullanılmaya başlanmıştır. Tekrar kullanılabilir birimler olarak sınıflar, nesneler ya da proje kapsamında geliştirilmiş bileşenler görülmüş, önceki projelerde hazırlanan bu birimlerin yeniden kullanılması sağlanmaya çalışılmıştır. Fakat zaman içinde yeterince tekrar kullanım amacına ulaşılamadığı ve

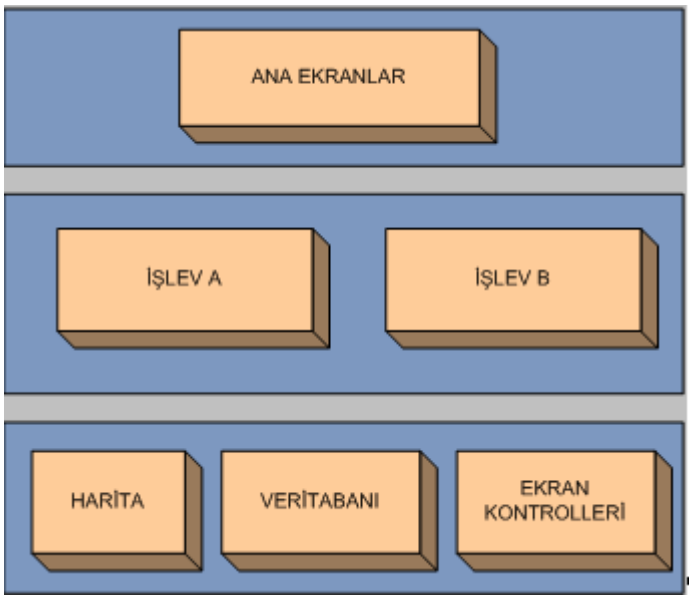
tekrar kullanılmak istenen birimlerde proje gereklerine göre değişiklik/iyileştirmelerin yapılması gerektiği görülmüş ve bu yüzden yeni bir arayış içine girilmiştir.

Şekil 1'de dikey bir yaklaşımla hazırlanmış bir yazılım gösterilmektedir. Tüm işlevler iç içe girmiştir, paketler birbirlerini herhangi bir bağımlılık kuralına uymaksızın kullanmaktadırlar. Mimaride herhangi bir katman ya da soyutlama söz konusu değildir.



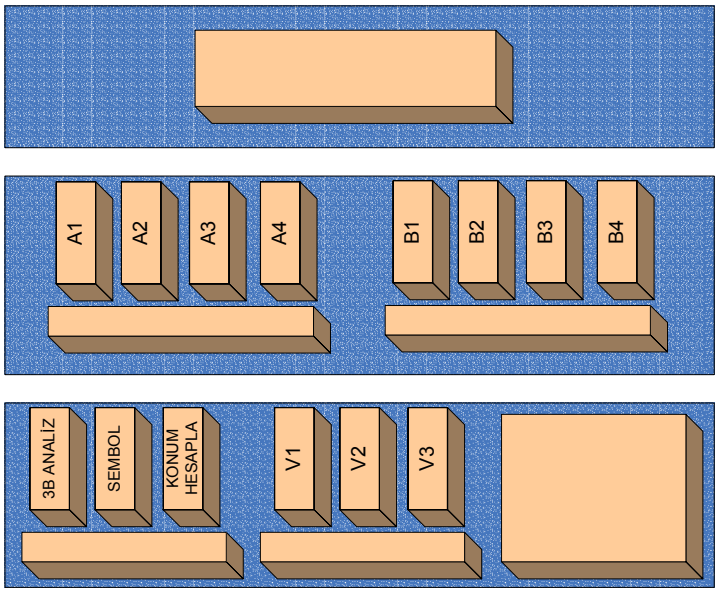
Şekil 1. Dikey Yaklaşımla Hazırlanan Yazılım Mimarisi

Projeler incelendiğinde yazılımların aslında kendini tekrarlayan yeteneklerden oluştuğu, her projede aynı yetenekleri sürekli olarak gerçeklemek yerine bu yetenekleri ortaklamanın çok daha etkin olacağı anlaşılmıştır. Bu amaçla yazılımlardaki ortak kısımlar incelenerek ortak bileşenler hazırlanmasına karar verilmiştir. Her bileşen tüm kullanıcılarının ihtiyaçlarını karşılamak üzere ayrı birer ürün olarak geliştirilmiştir. Bu yaklaşım sayesinde tekrar kullanılabilir birim olarak nesnelerden ortak kullanım için tasarlanmış bileşenlere geçilmiştir. Tüm projelerde bu bileşenler kullanılarak ihtiyaç duyulan yetenekler karşılanmıştır. Yeniden kullanılabilir bileşenlerin dış arayüzleri tanımlı ve kontrollü olduğundan daha önceleri kaçınılmaz olarak ortaya çıkan aşırı bağımlılık sorununu çözmek kolaylaşmıştır. Bağımlılık yönetimi, belirlenen katmanlar ve bu katmanları birleştiren arayüzler ile daha kolay uygulanmıştır. Şekil 2'de yatay yaklaşımla hazırlanmış, çeşitli katmanlardan oluşan ve katmanlarda bileşenlerin bulunduğu yazılım mimarisi gösterilmektedir. Farklı katmanlar içinde bulunan işlevler arasında soyutlama bulunmaktadır.



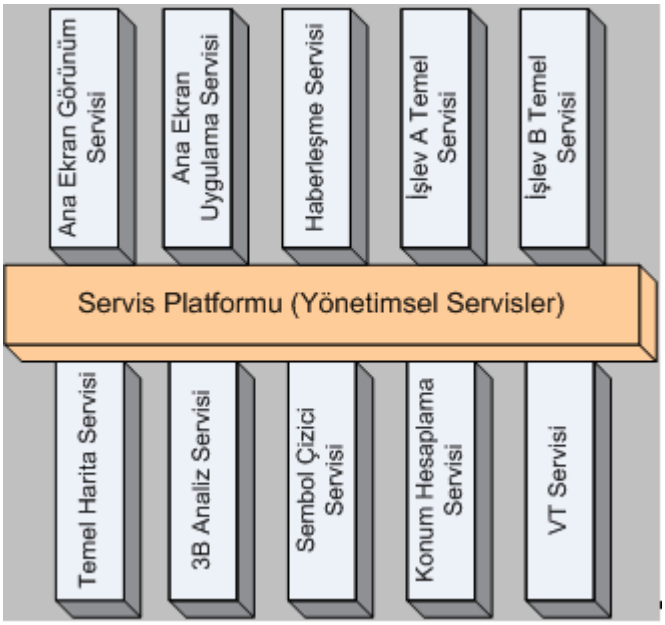
Şekil 2. Bileşenlerin Kullanıldığı Katmanlı Mimari

Fakat bu bileşenler zaman içinde her proje ihtiyacına cevap verme bakış açısıyla büyümüş, kontrolü ve idamesi zorlaşmaya başlamıştır. Bu da bileşenlerin projeler tarafından kullanımında zorluklara sebep olmuştur. Bu noktada ortak ve farklılaşan kısımları birbirinden ayırma ve eklenebilir kılma yaklaşımının gerekliliği görülmüştür. Alan analizi çalışmaları yapılmış, geliştirilecek yazılımların kapsam ve yetenek sınırları çizilmiştir. Böylece ortak yetenekleri barındıran "çerçeve" ve değişen yetenekleri barındıran "eklenti"lerden oluşan bir yaklaşıma geçilmesine karar verilmiştir. Çerçeve kullanımı ile bileşenlerin ortak yeteneklerinin belirli kurallar çerçevesinde kullanılması ve yine belirli kurallar dahilinde çerçevenin özelleşen proje ihtiyaçlarını karşılamasını sağlayacak eklentiler ile zenginleşmesi sağlanmıştır. Çerçeve her projede kullanılmış, ihtiyaçları karşılayamadığı durumlarda eklentiler hazırlanmıştır. Bu eklentiler zamanla bir havuzda toplanmış ve tekrar kullanımları ile projelere hızlı çözümler sağlanmıştır. Bunların da yetmediği durumlarda yeni eklenti geliştirme yoluna gidilmiş, ancak bu durumda yeni geliştirme maliyeti ortaya çıkmıştır. Bu yaklaşımla birlikte tekrar kullanılabilir birimlerin adı "çerçeveler ve eklentiler" olmuştur. Şekil 3'te çerçeveler ve eklentilerden oluşan bileşenler ve katmanlar gösterilmektedir. Bileşenler eklentiler sayesinde genişleyebilmektedir.



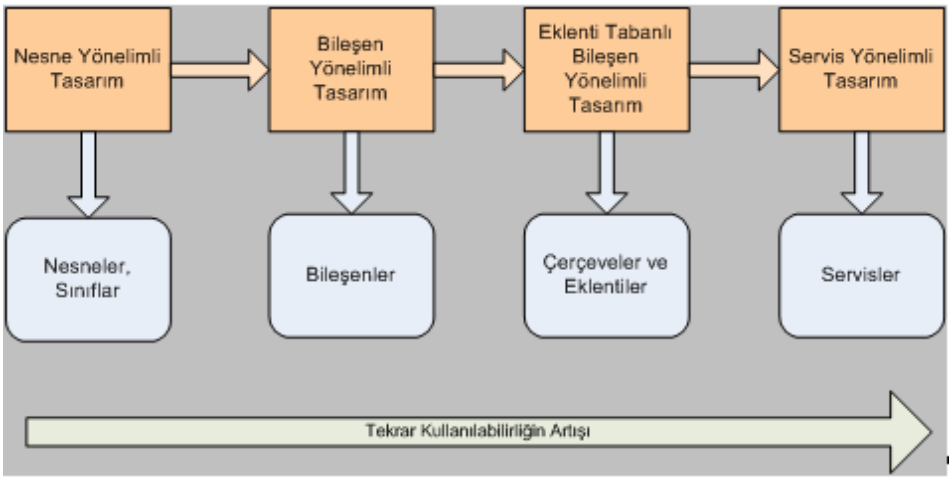
Şekil 3. Çerçeve ve Eklentilerden Oluşan Bileşenler ve Katmanlar

Çerçeve ve eklentilerden oluşan yazılımlar projelerin ihtiyaçlarını büyük oranda karşılamıştır. Fakat bazı projelerde bileşen ya da eklentilerin barındırdıkları yeteneklerden çok az bir kısmı kullanılmak istenmiş, bu durumda gereksiz yere tüm çerçeve ve eklenti projeye teslim edilmiştir. Eklentiler çerçevelerden bağımsız olarak kullanılamadığı için istenilen bir eklenti yeteneğiyle beraber çerçevede bulunan çok sayıda diğer yetenek de teslim edilmek durumunda kalınmıştır. Bunun yerine birbirinden bağımsız olarak kullanılabilen ve birbirlerini tamamlayan yetenekleri sunan yazılım parçalarının kullanımının daha faydalı olacağı değerlendirilmiştir. Böylece servis tabanlı mimari yaklaşımın kullanımının uygun olacağına ve tekrar kullanılabilir birimler olarak da servislerin görülmesine karar verilmiştir. Yapılan alan analizi çalışmaları ile servislerin büyüklük ve kapsamları belirlenmiştir. Böylece servisler birbirleri ile zayıf bağımlı, taşınabilir, beraber çalışabilir, yüksek uyumlu, kendi içinde bütün yapılar olarak tasarlanmıştır. Bölüm 3'te görev yazılımlarında servis tabanlı mimarilerin kullanımı detaylı olarak verilmektedir. Şekil 4'te servisler ve servislerin birbirleri ile nasıl etkileştikleri gösterilmektedir. Her servis, servis platformunun yeteneklerini kullanarak ihtiyaç duyduğu servislere ulaşabilmekte ve kullanabilmektedir. Servis platformu yönetimsel servisleri içeren altyapıdır ve servislerin birbirlerine standart bir şekilde ulaşmalarını ve birbirlerini kullanmalarını sağlamaktadır. Örneğin Şekil 3'te belirtilen eklenti tabanlı mimaride, Ana Ekran Bileşeninin sembolleri çizebilmek için Temel Harita Bileşeni ile birlikte Sembol Çizici Eklentiyi kullanması gerekmekteyken, servis yaklaşımına geçiş ile Sembol Çizici Eklentisi yerine Şekil 4'te gösterilen Sembol Çizici Servisi hazırlanmış ve bu servisin diğer servisler tarafından doğrudan kullanılmasına olanak sağlanmıştır.



Şekil 4. Servisler ve Etkileşimleri

Şekil 5'te ASELSAN görev yazılımlarında kullanılan uygulama geliştirme yaklaşımının zaman içindeki değişimi gösterilmektedir. En üst sırada yer alan dikdörtgenler ile kullanılan yaklaşımlar tanımlanmış, her yaklaşımın altında tekrar kullanılabilir yazılım birimleri verilmiştir. Soldan sağa doğru gidildikçe tekrar kullanılabilirlik artmıştır. Bu da projelerin gittikçe azalan sürelerde ve daha kaliteli biçimde teslim edilmelerini sağlayan önemli bir etmen olmuştur.



Şekil 5. ASELSAN Görev Yazılımlarında Kullanılan Yazılım Mimarilerinin Evrimi

3 Görev Yazılımlarında Servis Tabanlı Mimariler

ASELSAN görev yazılımlarında servis tabanlı mimarilerin doğrudan kullanımına geçişin kolay olmayacağı ve eski projelerin de desteklenmesi gerektiği için sıkıntılar oluşabileceği değerlendirilmiştir. Bu nedenle mevcut bileşenlerin servisler haline getirileceği, yeni geliştirilecek bileşenlerin ise servisler olarak tasarlanacağı aşamalı bir geçiş planlanmıştır. Böylece mevcut bileşenler hem eski projelerde kullanılabilecek hem de üzerlerine yazılacak bir kabuk sayesinde servis olarak da hizmet verebilecek hale geleceklerdir. "Migrating to a service-oriented architecture" isimli makaleden yola çıkılarak bu aşamalı geçiş planlanacak, geçmiş projelere destek devam ederken gelecek projelere de çözüm sağlanacaktır [2].

Servis tabanlı mimarilere geçişi tetikleyen ihtiyaçlar şu şekilde sıralanabilir:

- Yazılım birimlerinin çalışma zamanı yönetimi gereklidir. Yazılım birimlerinin çalışma zamanında kullanılmayan yeteneklerin kapatılması, ihtiyaç duyulduğunda tekrar yeteneklerin kullanıma alınması, bu sayede kaynak durumuna, kullanıcı haklarına göre yazılımların yapılandırılması sağlanacaktır.
- Yazılım birimlerinin konfigürasyon yönetimi gereklidir. Yazılım birimlerinin konfigürasyonlarının uyumlu oldukları birimlerle birlikte takip edilmesi gerekir. Çerçeve ve eklenti yaklaşımında her çerçeve ve uyumlu eklentilerin farklı proje ihtiyaçları ile zenginleşmesi ve sıkı bağımlılıkları nedeniyle tüm birimlerin sürümlerinin karşılıklı olarak takibi gereği ortaya çıkmıştır. Bu yüzden sürüm takibinde zorluklar yaşanmıştır.
- Yazılım birimlerinin ihtiyaç duyulan yetenekleri kapsaması gereklidir. Çerçeve ve eklenti yaklaşımında, çerçeveye ihtiyaç duymayan, sadece bir

eklentinin yeteneklerini kullanmak isteyen yazılımlar kullanmasa da çerçeve ve eklentiyi beraber almak durumunda kalmıştır. Eklenti, çerçeve olmadan çalışabilen bir yazılım birimi değildir. Bu yüzden çerçeve ve her bir eklenti ayrı ve bağımsız çalışabilen servisler haline getirilerek kullanım zorluğunun çözüleceği değerlendirilmiştir.

- Yazılım birimlerine erişimin ve yetenekleri kullanmanın standartlaştırılması gereklidir. Çok sayıda yazılım biriminin olduğu ve her birimin ayrı teknoloji kullanarak yeteneklerini sunduğu bir ortamda bu birimleri kullanan bir yazılımı hazırlamak kolay olmamaktadır. Birimlerin haberleşme, veri protokolleri ile kullanılan teknolojiler farklılaşabilmektedir. Standart bir servis altyapısında ortak haberleşme protokolleri kullanımıyla uyumluluk ve birlikte çalışabilirlik sorunlarının ortadan kaldırılacağı değerlendirilmektedir.
- Yazılım ürün hattı yaklaşımını destekleyecek, yazılım birimlerinin koreografisini sağlayacak bir altyapı ihtiyacı bulunmaktadır. Bu altyapının etkin kullanımını sağlayacak bir mimari geliştirme ihtiyacı vardır. Yazılım birimlerini kullanma ve kontrol etme yönteminin ortaklaştırılması ile uyumlu çalışma, esneklik, bakım yapılabilirlik hedeflerine ulaşmak için servis tabanlı mimarinin en uygun çözüm olduğu değerlendirilmiştir.
- Çerçeve ve eklenti yaklaşımıyla tekrar kullanılacak yazılım birimlerinin sınırlarını çizmek güç olmaktadır. Birimler arasındaki statik bağımlılıklar nedeniyle alınmak istenen bir yetenek beraberinde pek çok yeteneği de sürüklemektedir. Servis yaklaşımıyla bağımlılıkların hafifleyeceği, arayüzlerin dinamik olarak kurulabileceği değerlendirilmektedir. Böylece istenmeyen yetenekleri alma zorunluluğu ortadan kalkacaktır.

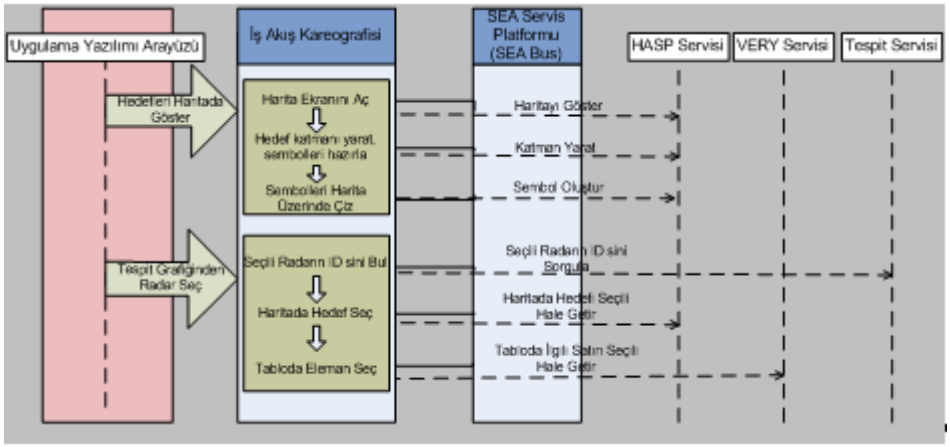
Servisler bileşenlere göre daha küçük, kendi başına tanımlı işlevleri yerine getirebilen birimler olacaktır. Mevcut büyük bileşenlerin parçalanmadan üzerlerine kabuk yazılarak servis haline getirilmesi planlanmıştır. Bu geçiş aşaması için ara bir çözüm olarak uygulanacaktır. Yürümekte olan çok sayıda projeye desteği sürdürebilmek, gelebilecek değişiklik isteklerine, eklemelere hızlı bir şekilde cevap verebilmek ve var olan bileşenlerden servis olarak en az emekle faydalanmak ana hedeftir. Henüz geliştirilmemiş olan yazılım birimlerinin servis bakış açısıyla tasarlanması, mevcut büyük bileşenlerin ise zaman içinde bölünerek servislere dönüştürülmesi kararlaştırılmıştır. Böylece yürümekte olan çok sayıda projeye desteği sürdürmek ve gelebilecek değişiklik isteklerine ve eklemelere hızlı bir şekilde cevap vermek mümkün olacaktır.

Servislerin sunulması, yönetilmesi ve sorgulanabilmesi için bir servis platformu gereklidir. Bu amaçla ASELSAN Görev Yazılımları için "Servis Erişim Altyapısı (SEA)" platformu belirtimi tanımlanmıştır. SEA platformu, servislerin, fiziksel yerleri ve detaylarının bilinme ihtiyacı olmadan erişilebilmesini sağlayan, erişim kontrolünü ve bağımlılıklarını yöneten bir altyapı sunar. SEA, OSGI standardının [3] sadece ihtiyaç duyulan servislerinin gerçekleştirildiği bir platform olarak algılanabilir. Bunun dışında servis yönelimli mimarilerin tasarlanabileceği ve kullanılabileceği altyapılar sunan ticari ürünler de mevcuttur. Bu ürünlerden "Visual Studio Team System" ürün ailesi sağladığı çeşitli araçlar ile çözüm sunabilmektedir.

4 Servis Tabanlı Mimari Kullanımına Geçiş İle Hedeflenen Kazanımlar/Karşılaşılabilecek Sorunlar

Servis yaklaşımı bir projeyi oluşturmak amacıyla kullanılacak servislerin belirlenmesi faaliyetini kapsar. Uygulama geliştirmek bu servislerin belli bir koreografi içinde kullanılması, belirli bir düzende çağrılması ve yönetilmesi faaliyetleri olarak tanımlanabilir. Bu faaliyet [4]'de belirtilen "İş Akış Koreografisi" olarak adlandırılmaktadır. Koreografiyi yapan birimler birer iş servisi de olabilir. Bu birimler diğer temel servisleri kullanarak işe yönelik yetenekleri gerçeklerler. İş servisleri içinde çağrılan diğer servislerde hata oluşması durumunda geri alma yapılır. İş servisleri birer işlemdir ve işlem yönetim mekanizması olmalıdır.

Şekil 6'da örnek bir uygulama yazılımı geliştirme faaliyeti gösterilmektedir. Ortamda genel servisler bulunmaktadır. Bu servisler birbirlerinden bağımsız olarak yeteneklerini sunarlar. Servis platformu servislerin birbirlerini bulmalarını sağlayan bir altyapı olarak görev yapar. Uygulama yazılımı genel servisleri ve iş servislerini kullanarak hazırlanacaktır.



Şekil 6. Uygulama Yazılımı Hazırlama Yaklaşımı

Bu servisleri bir araya getirerek yazılım geliştirmek standart ve belirli kurallarla tanımlı bir hale getirilecektir. Servis platformu sayesinde istenilen arayüzü gerçekleyen servisleri aramak, bulmak ve kullanmak standartlaştırılacaktır.

Yazılım birimleri kolay test edilebilir olacaktır. Servisler tanımlı arayüzler üzerinden kullanıldıkları için test edilebilmeleri kolaylaşacaktır. Her servis için sahte gerçeklemeler yazılarak istenilen yazılım birimleri başka birimlerin sadece sahteleriyle test edilebilir kılınacaktır. Ayrıca servislerin tekrar test edilmemesi, sadece arayüz testleri yapılması, proje bazında test edilmiş bileşenlerin kullanılması sayesinde hata sayısının azaltılması hedeflenmiştir.

Yazılım ürün hattı yaklaşımına geçişi destekleyecek bir altyapı kurulması sağlanacaktır. Yazılım ürün ailelerinden projelerde kullanılacak özel ürünler tanımlı yöntemlerle, servis altyapısı üzerinde belirli kurallar çerçevesinde hazırlanacaktır.

Hedeflenen kazanımların yanında servis tabanlı mimarilere geçişle birlikte performans sıkıntıları, konfigürasyon yönetiminin zorlaşması, servis geliştirme güçlüğü gibi sorunların da ortaya çıkabileceği değerlendirilmiştir.

5 Sonuç

ASELSAN görev yazılımlarında kullanılan mimarilerin gelişmesinde en önemli etkenler olarak tekrar kullanılabilirliği artırmak, karmaşıklığı azaltmak, daha hızlı ve kaliteli yazılımları daha kısa sürede geliştirmek, ürün ve yetenek çeşitliliğini yönetmek, yazılım ürün hattı yaklaşımını desteklemek ve bakım yapılabilirliği artırmak ihtiyaçları önemli rol oynamıştır. Bu hedefler, mimarinin nesne yönelimli yaklaşımlardan bileşenlere, bileşenlerden çerçeve ve eklentilere, çerçeve ve eklentilerden servis yaklaşımına geçişi tetikleyen etkenler olmuştur.

Alan analizi çalışmalarının desteğiyle büyük yazılımların daha kolay yönetilebilen küçük servislere ayrıştırılması amaçlanmıştır. Bu aşamada, farklılaşan ürün ihtiyaçlarını karşılayacak servislerin hazırlanmasının yeterli olmayacağı, servislerin üzerinde çalışabileceği bir platformun oluşturulması gereği de ortaya çıkmıştır. Bu amaçla çok sayıda servisin uyumlu bir şekilde çalışarak iş akışını gerçekleştirmelerini sağlayacak, hem bir belirtim hem de gerçekleme olan SEA platformu hazırlanmaktadır. Platform, servislerin beraber çalışmalarını kolaylaştırma ve erişim yöntemlerini standartlaştırma hedeflerini taşımaktadır. SEA platformunun tasarımında 4. bölümde belirtilen kazanım hedefleri ve olası sorunların çözümleri göz önüne alınmıştır. Performansı artırmak amacıyla servis platformunda her bir servise erişim yöntemi servisin bir nesne olarak alınması ve sunduğu yeteneklerin doğrudan metot çağrımı ile kullanılması şeklinde tasarlanmıştır. Servislerin uyumlu oldukları diğer servis sürümleriyle birlikte servis platformuna kaydedilmesiyle birlikte çalışabilir sürümlerin takibi kolaylaştırılacaktır. Servis platformu sayesinde servislere erişim yönteminin standartlaştırılması, yeniden kullanımın ve bakım yapılabilirliğin artırılması hedeflenmiş, servis birimlerinin testlerinin tekrarlanması yerine sistem entegrasyon testlerinin yeterli olacağı değerlendirilmiştir.

Kaynakça

1. Maer E., Jenny A., Ali A., Sook C., Philippe C., Pal K., Min L. Tony N., “Patterns: Service-Oriented Architecture and Web Services”, 2004
2. Kishore C., Kerrie H., Edward T., “Migrating to a service-oriented architecture”, 2003
3. www.osgi.org.
4. Olaf Z., Pal K., Clive G., “Elements of Service-Oriented Analysis and Design”, 2004

Gerçek Zamanlı Gömülü Yazılım Geliştirmede Bileşen Entegrasyon Deneyimleri

Evrim Kahraman[1], Tolga İpek[2]

[1,2] ASELSAN A.Ş. Savunma Sistem Teknolojileri Grubu
[1] ekahraman@aselsan.com.tr, [2] tipek@aselsan.com.tr

Özet. Bu bildiride, ASELSAN A.Ş. Savunma Sistem Teknolojileri (SST) Grubunun gerçek zamanlı gömülü yazılım projeleri kapsamında gerçekleştirilen bileşen entegrasyonu yaklaşımlarından bahsedilecektir. Bu bağlamda, yazılım içi bileşen entegrasyonu için, yazılım geliştirmede kullanılan "*Unified Modelling Language*" (*UML*) modelleme dilinin sunduğu alt yapılardan("*port", "interface class"*) ve yazılım tasarımında benimsenen "*yayımcı-abone*" ("*publish-subscribe*") kalıbından bahsedilecektir. Yazılım içi bileşen entegrasyonu için gerçekleştirilen yazılım tasarımlarının, yazılım dışı bileşenlerin entegrasyonu durumu söz konusu olduğunda, arakatman teknolojileri ile birlikte kullanım yöntemine de değinilecektir.

1 Giriş

1980'li yılların başında, yapısal yazılım geliştirme yerini nesne yönelimli programlama ve tasarım gibi kavramlara bırakmaya başlamıştır. Devam eden süreçte, yeniden kullanılabilirlik, taşınabilirlik gibi kaygılarla, bileşenlerden oluşan bir yazılım kavramı ön plana çıkmıştır. Bileşen, bir görevi yerine getirmek üzere gerekli verileri tutan, yetenekleri ve dış dünyaya sunacağı servisleri belirli olan yazılım parçası olarak düşünülebilir. Pek çok bileşenden oluşan bir yazılım tasarımında, önemli noktalardan biri bileşenlerin sınırlarının çizilmesi ve görevlerinin ayrıştırılmasıdır. Bir diğer önemli nokta ise bileşenler arası entegrasyonun yani haberleşmesinin nasıl sağlanacağıdır. Bileşenler arası haberleşme için işletim sisteminin sağladığı altyapılar, kullanılan yazılım geliştirme ortamı ya da dilinin sağladığı yetenekler kullanılabilmektedir. Ayrıca bileşenlerin haberleşmesine yönelik "yayımcı-abone"[1] gibi tasarım kalıpları da mevcuttur. Dağıtık ve farklı platformlarda çalışan yazılımların içerisinde yer alan bileşenlerin haberleşmesi ihtiyacına yönelik olarak arakatman teknolojileri olarak bilenen bir takım standartlaşma çalışmaları, OMG[2] tarafından yayınlanan CORBA [3] gibi, dünya çapında yürütülmektedir.

Bu bildiride, bileşen ve bileşen entegrasyonuna yönelik temel kavramlara değinilecek, ve ASELSAN SST Grubu gerçek zamanlı gömülü yazılım projelerinde[4][5] uygulanan bileşen entegrasyon deneyimlerinden bahsedilecektir. Bahsi geçen projelerde, bileşen entegrasyonuna yönelik "yayımcı-abone" tasarımından, UML[6] modelleme dilinin sunduğu altyapıların nasıl kullanıldığından ve dağıtık yazılımlarda çalışan bileşenlerin entegrasyonu için CORBA arakatman teknolojisi kullanılarak nasıl uyumlandırılabildiğinden bahsedilecektir.

2 Bileşen ve Bileşen Entegrasyonu

Bileşen, 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 ve başka bileşenlerle entegre edilebilmelidir. Bileşen, iç tasarımı ve detayları bilinmeden sadece tanımlanmış servis arayüzleri aracılığıyla kullanılabilmelidir. [7]

Bileşen, tek bir obje ya da obje grubu olarak tasarlanabilir. Yeniden kullanıma maksimum seviyede hizmet edebilmesi için, bileşenin içindeki parçaların ilişkileri yüksek ("*high cohesion*"), bileşenin diğer bileşenlere bağımlılığı ise zayıf olmalıdır. ("*loosely coupled*")

Bir bileşenin, diğer bileşenlerle entegre edilebilmesi için tanımlanması gerekenler şu şekilde sıralanabilir;

- Bileşenin diğer bileşenlere sağlayacağı servisler
- Bileşenin diğer bileşenlerden talep edeceği servisler
- Bileşenin diğer bileşenlerle haberleşme metodu

Farklı yazılımlarda çalışan bileşenlerin entegrasyonu için işletim sisteminin sunduğu soket, paylaşım alanı ("*shared memory*"), mesaj kuyruğu ("*message queue*") gibi haberleşme mekanizmaları kullanılabilirken, yazılım içi bileşenlerin haberleşmesinde, yazılım tasarımında kullanılan ortamın ya da yazılım geliştirmede kullanılan programlama dilinin sunduğu daha soyut mekanizmalar da kullanılabilir. Örneğin, UML modelleme dili, "*port*" ve "*interface class*" tanımlarıyla buna yönelik altyapılar sunmaktadır. Sistem tasarımında, sistemin beklenilen görevleri yerine getirilebilmesi için fiziksel olarak ayrılmış birden fazla yazılımın tanımlanması ihtiyacı oluşabilir. Bu ihtiyaç, performans kaygılarından, altyapı kısıtlarından, mevcut sistem ya da yazılımların kullanımı gibi durumlardan kaynaklanabilir. Üstelik bu yazılımların çalışacakları platformlar birbirinden farklılık da gösteriyor olabilir. Hatta, işlemci yüküne göre, bileşenin çalışacağı platformun çalışma zamanında dinamik olarak belirlenmesi de gerekebilir. Farklı yazılımlarda çalışan bileşenlerin entegrasyonu için belirtilen bu ihtiyaçları çözmek üzere standartlaşmış, CORBA gibi, arakatman teknolojileri mevcuttur.

Bileşenler arası servis paylaşımı için, servisin çağırılmasının ile entegrasyon sağlanırken, bileşenler arası veri haberleşmesi için, ek olarak, verinin değişmesi durumunda, diğer bileşenlere haber verilmesi ihtiyacı olabilmektedir. Bu yaygın duruma yönelik tasarım kalıpları mevcuttur. Bunlardan en bilineni "yayımcı-abone" tasarım kalıbıdır.

3 Aselsan Deneyimleri

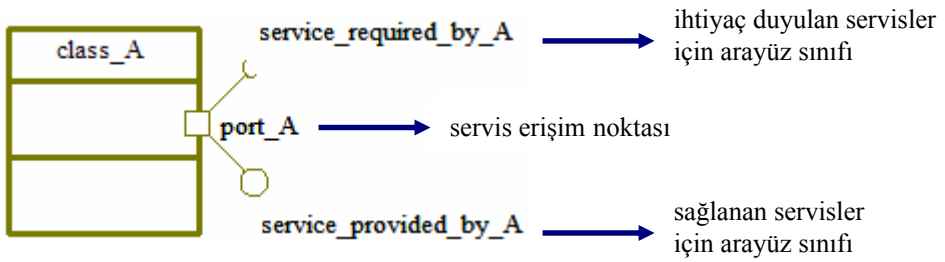
Bu bölümde, Aselsan SST Grubunun, model tabanlı yaklaşım ile geliştirilen, gerçek zamanlı gömülü yazılımlarında, bileşen entegrasyonu kapsamında yapılan çalışmalardan bahsedilecektir. UML modelleme dili ile modellenen yazılımların,

bileşen entegrasyon seviyeleri projeden projeye farklılık gösterebilmektedir. Bu bağlamda, yazılım içi, yani model seviyesinde gerçekleştirilen entegrasyon çalışmaları ve sistem seviyesinde gerçekleştirilen entegrasyon çalışmaları ele alınacaktır.

3.1 Model Seviyesi Bileşen Entegrasyonu – Veri Dağıtımı

Yazılım bileşenleri, kendilerinden beklenilen görevleri yerine getirmek üzere pek çok veri tutmaktadırlar. Bunların bir kısmı bileşenin iç kullanımına yönelikken, bir kısmı diğer bileşenler tarafından ihtiyaç duyulacak veriler olabilmektedir. Bu bölümde anlatılacak yöntemde, veri paylaşımı ve abonelik mekanizmalarının çalışma zamanında kurulması hedeflenmiştir.

UML2.0'da, sınıfların, sağladıkları servisler ve talep ettikleri servisler, arayüz sınıfları ("*Interface Class*") içerisinde tariflenmektedir. "*Port"*, bu servislere erişimin sağlandığı, bileşenin servis erişim noktasıdır. Bileşenler birden fazla servis erişim noktasına ve bunlar üzerinden verdikleri farklı servislere sahip olabilirler. Şekil 1'de örnek bir sınıf üzerinde erişim noktası ve servisler gösterilmektedir.



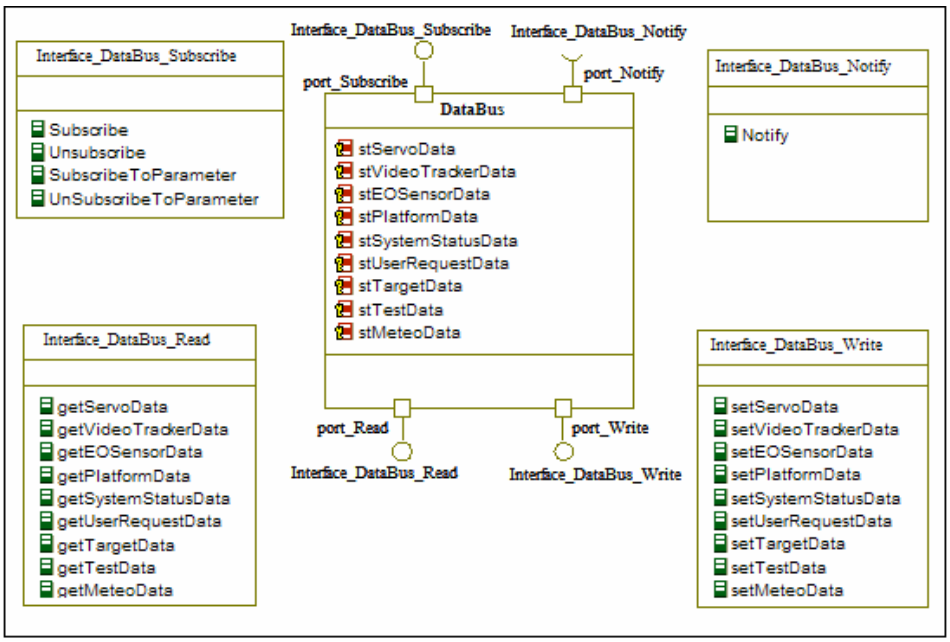
Şekil 1. Sınıf, servis erişim noktası ve arayüz sınıfları gösterimi

"yayımcı-abone" tasarım kalıbı içerisinde yer alacak bir bileşenin sahip olması gereken arayüz sınıfları ve bu arayüzler üzerinden sağlayacağı servisler "A" verisi örneği için Tablo 1'de verildiği gibi belirlenmiştir.

Tablo 1. "yayımcı-abone" tasarım kalıbı ile uyumlu arayüz sınıfları ve servisler

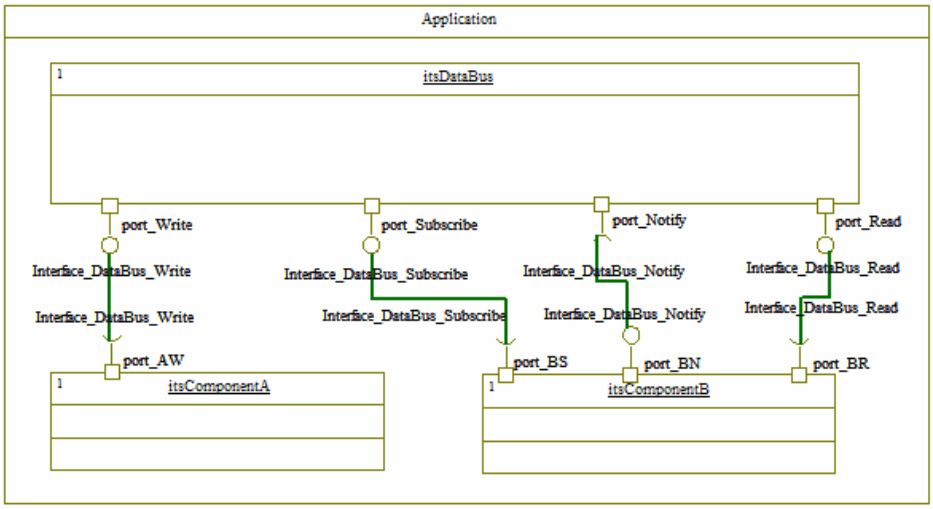
Arayüz Sınıfı	**Servisler**
Interface_DataBus_Subscribe	subcribe(),unsubscribe()
Interface_DataBus_Read	getA()
Interface_DataBus_Write	setA()
Interface_DataBus_Notify	notify()

Sınıflar arası veri paylaşım ihtiyacının çokluğunun karmaşık ilişkiler doğurmasını engellemek için, paylaşılması gereken üst seviye verilerin, merkezi bir sınıfta toplanması yöntemi benimsenmiştir. "*DataBus*" olarak isimlendirilen merkezi sınıf, "yayımcı-abone" tasarım kalıbının ifade ettiği gibi çalışacak ve bileşenler arası veri haberleşmesini sağlayacaktır. Şekil 2'de, "DataBus" sınıfına erişim noktaları, arayüz sınıfları ve üzerindeki servisler görülmektedir.

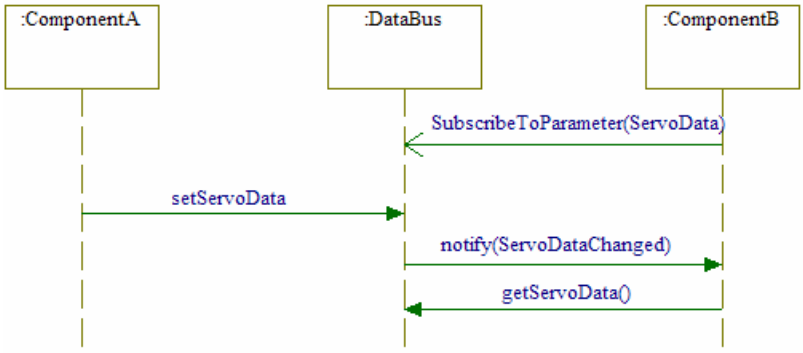


Şekil 2. "DataBus" sınıfı servis erişim noktaları, arayüz sınıfları ve üzerindeki servisler

Veri sınıfına, servis erişim noktaları üzerinden bağlanan bileşenler, verinin değerini okuyabilmekte, değiştirebilmekte ve ilgilendikleri verilere abone olarak değişimlerden haberdar olabilmektedirler. Şekil 3 ve Şekil 4'te sırasıyla, iki bileşenin "DataBus" üzerinden veri paylaşımına yönelik bağlantısı ve veri akış diyagramı verilmektedir.



Şekil 3. DataBus ve diğer bileşenlerin bağlantısı



Şekil 4. Örnek veri akışı

Merkezi bir veri sınıfı tasarımının bileşenler arası ilişkilerin karmaşıklığını engellemesi temel avantajı olarak gözükmektedir. Ayrıca, verilerin toplu olarak bir yapıda tutulması, bu verilerin tümünün ya da alt gruplarının silinmez hafızada ya da dosyada saklanmasını kolaylaştırmaktadır. Bunun yanında ortak fonksiyonların - kayıt alma gibi

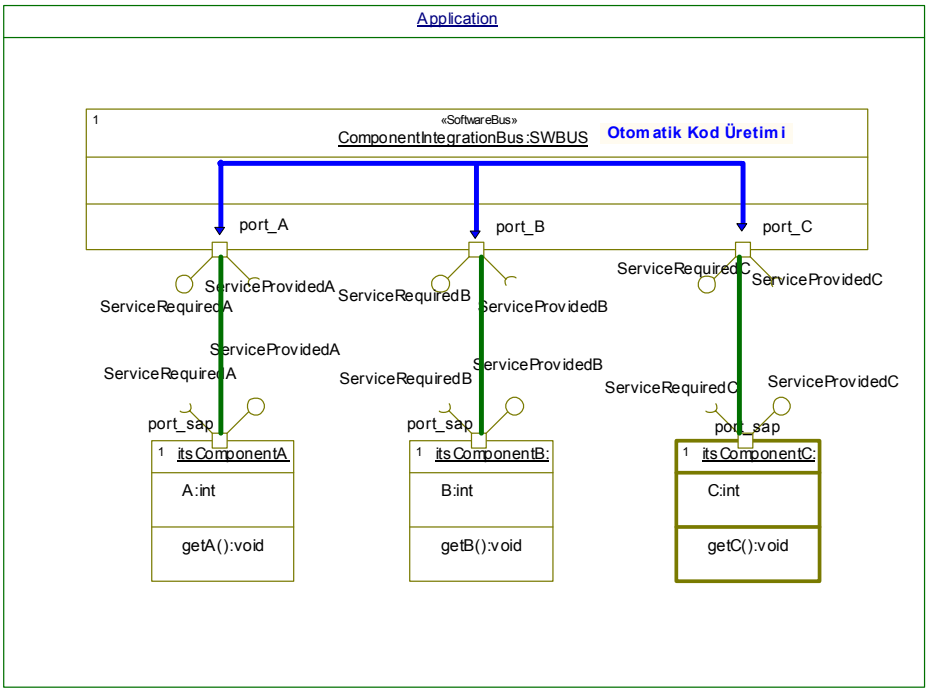
tanımlanmasına da imkan vermektedir. Bu yöntemin en büyük dezavantajı veri sınıfının farklı projelerde yeniden kullanılabilirliğinin az olmasıdır. Ayrıca merkezi veri sınıfındaki değişikliklerden, veri sınıfına bağlanan tüm bileşenler etkilenmektedir.

3.2 Model Seviyesi Bileşen Entegrasyonu – Otomatik Servis Paylaşımı

Model seviyesi otomatik servis paylaşımı yönteminde bileşenlerin UML 2.0'da tanımlanmış arayüzleri kullanılarak, derleme işlemi öncesinde, model seviyesinde bileşenlerin otomatik olarak entegre edilmesi hedeflenmiştir.

Uygulama bir bütün olarak düşünüldüğünde, uygulamayı oluşturan bileşenlerin talep ettikleri servisleri veren mutlaka bir başka bileşen olmalıdır. Bu, yazılım tasarımında göz önünde bulundurulur. Dolayısıyla daha tasarım aşamasında hangi bileşenin istediği servisi hangi bileşenden alacağı bellidir. Model seviyesinde otomatik servis paylaşımı yazılım tasarımında verilen bu kararların, yazılımın kodlanması zamanında yazılım işçiliği gerektirmeden ve yazılımın çalışma zamanında minimum işlemci gücü kullanarak, otomatik olarak gerçekleştirilmesini sağlamaktadır.

Model seviyesinde otomatik servis entegrasyonunda düşünülen çözümün yazılım modelinde gösterimi Şekil 5'te verilmektedir.



Şekil 5. Model seviyesi otomatik servis paylaşımı

Şekilde görülen ve bileşenlerin entegrasyonunu sağlayan sınıf her bir bileşen ile kendi arayüzü tanımları ile haberleşecek bir sınıftır ve bu yazının kalan kısmında "SWBUS" olarak adlandırılacaktır.

SWBUS sınıfı şekildeki gibi modellendiğinde, diğer tüm bileşenlerin sağladığı ve ihtiyaç duyduğu servis bilgilerine sahip olmaktadır. Bundan sonra yapılması gereken bu arayüzlerin analiz edilmesi ve ihtiyaç duyulan ve sağlanan servislerin eşleştirilmesidir (bakınız Tablo 2). Bu işlem yazılan bir script ile otomatik olarak yapılabilmekte ve analiz sonrasında bileşenleri entegre eden kod otomatik olarak üretilebilmektedir. Analiz aşamasında üretilen bir bilgi de ihtiyaç duyulan tüm servislerin karşılanıp karşılanmadığıdır ki, bu bir yönden tasarımın doğrulanması işlemidir.

Tablo 2. Arayüz sınıfları ve servisleri

Arayüz Sınıfı	Servisler
ServiceProvidedA	int getA()
ServiceProvidedB	int getB()
ServiceProvidedC	int getC()
ServiceRequiredA	int getB()
ServiceRequiredB	int getC()
ServiceRequiredC	int getA()

SWBUS sınıfının bileşen entegrasyonunu sağlarken herhangi bir kayıtlanma, veri tabanı veya yeni bir mesaj kuyruğu kullanmaması minimum işlemci gücü harcamasında önemli bir etkendir. Uygulamanın ve bileşenlerin gerçek zamanlılık gereklerinin etkilenmemesi, SWBUS'ın çalışmasının tamamen kendisini kullanan bileşenlerin öncelikleri ile olması ile sağlanabilmektedir. Bu yöntemde fazladan işlemci gücü sadece otomatik üretilen kodun içerdiği anahtarlama kodları için harcanmaktadır.

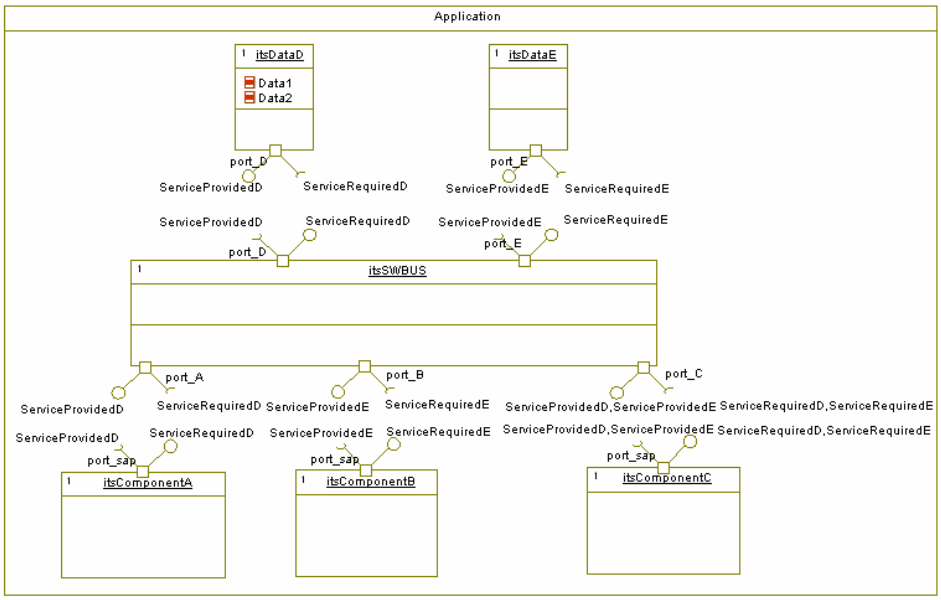
Model seviyesi bileşen entegrasyonunun en önemli dezavantajı tasarım zamanında belli olmayan bir bileşenin yazılıma sonradan, çalışma zamanında eklenmesinin mümkün olmamasıdır. Başka bir dezavantajı da yazılım model diyagramında hangi bileşenin hangi servisi kimden aldığı bilgisinin görülmemesidir.

3.3 Model Seviyesi Bileşen Entegrasyonu – Otomatik Veri Dağıtımı

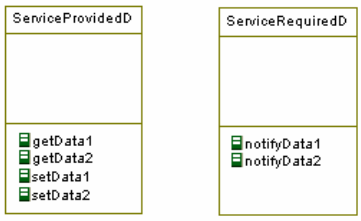
Önceki bölümlerde anlatılan, veri dağıtım yöntemi ile otomatik servis paylaşımı yöntemi birleştirilerek, derleme zamanında veri dağıtımının otomatik sağlandığı bir yöntem hedeflenmiştir. Derleme zamanı otomatik servis paylaşımı yönteminin, verinin değerinin okunması ("*get*") ve yazılması ("s*et*") servisleri için kullanılabileceği noktasından hareketle, bahsi geçen iki yöntemin birleştirilebileceği değerlendirilmiştir. Veri paylaşımı için kullanılmak istenen "yayımcı-abone" tasarım kalıbının gerektirdiği

abonelik (“*subscribe*”) ve bilgilendirme (“*notify*”) servislerinin de otomatik servis dağıtımı yeteneğine eklenmesi sağlanarak, otomatik veri dağıtımı gerçekleştirilmiştir.

Veri dağıtım yönteminde benimsenen merkezi veri sınıfı yaklaşımının tespit edilen dezavantajları da bu çalışma kapsamında giderilmek istenmiştir. Merkezi bir veri sınıfının yeniden kullanılabilirlik açısından dezavantajı dikkate alınarak, daha küçük boyutta veri anlamlı gruplarını içeren bir tasarım yapılmıştır. Veri sınıflarının arayüzlerinde, “yayımcı-abone” tasarım kalıbındaki servisler tanımlanmış ve bu servislerin otomatik servis paylaşımı ile diğer bileşenlerle paylaşımı sağlanmıştır. Ayrıca merkezi bir veri sıfı ve ona bağlanan bileşenler durumunda, merkezi veri sınıfındaki değişimden tüm bileşenler etkilenmektedir, oysa ki daha küçük veri grupları tanımlandığında değişikliğin etki alanı küçülecek ve sadece ilgili bileşenler etkilenecektir. Model seviyesi otomatik veri paylaşımı nesne model diyagramı Şekil 6’da, veri sınıflarının servisleri Şekil 7’de gösterilmektedir.



Şekil 6. Model seviyesi otomatik veri paylaşımı



Şekil 7. Veri sınıfı servisleri

3.4 Sistem Seviyesi Bileşen Entegrasyonu

Sistem seviyesi, yani yazılım dışı bileşenlerin haberleşmesine yönelik olarak yukarıda bahsi geçen çözümlerinde tümünde CORBA kullanılabilir. Şekil 8'de, CORBA üzerinden, çalışma zamanı abonelik ile veri paylaşımının sağlanmasına yönelik model yer almaktadır. CORBA yeteneklerini sağlayan bileşen herhangi bir diğer bileşen gibi ilgili sınıfa bağlanmakta ve diğer yazılım ile bağlantıyı sağlayabilmektedir.

Şekil 8. Veri sınıfı servisleri

CORBA ile standart TCP soket performansı karşılaştırıldığında, CORBA'nın performans açısından herhangi bir problem yaratmadığı görülmüştür. Üstelik, kodlama anlamında, mesaj kodlayıcı yazılmasına gerek bırakmıyor olması ile de avantajlıdır.

4 Sonuç

Bu bildiride ASELSAN SST Grubu gerçek zamanlı gömülü yazılımları kapsamında yürütülen bileşen entegrasyonu çalışmalarından bahsedilmiştir. Çalışmanın başlangıç

aşamasında farklı ekipler tarafından, yazılım içi bileşen entegrasyonuna yönelik olarak aşağıdaki iki çözüm ortaya konulmuştur;

- **Çalışma zamanı "yayımcı abone" kalıbı":** UML 2.0 servis erişim noktaları ve servis tanımları üzerinden "yayımcı-abone" kalıbı kullanımıyla veri haberleşmesinin sağlanması verinin değerinin okunması, değiştirilmesi ile birlikte çalışma zamanında veriye abone olunması sağlanabilmekte ve verinin değerinin değişmesi durumunda abone olan bileşenlere bilgi verilmektedir.
- **Derleme zamanı otomatik servis paylaşımı:** Servis sağlayan ve servis isteyen bileşenlerin arasına eklenen ve kodu otomatik üretilen merkezi bir sınıf üzerinden bileşenlerin servislerinin bağlanması servis bağlantılarının kara-kutu yaklaşımı ile derleme zamanı öncesinde merkezi bir noktadan yapılması sağlanmaktadır.

Derleme zamanı otomatik servis paylaşımı yönteminin, verinin değerinin okunması ("*get*") ve yazılması("s*et*") servisleri için kullanılabileceği noktasından hareketle, bahsi geçen iki yöntemin birleştirilebileceği değerlendirilmiştir. Veri paylaşımı için kullanılmak istenen "yayımcı-abone" tasarım kalıbının gerektirdiği abonelik ("*subscribe*") ve bilgilendirme ("*notify*") servislerinin de otomatik servis dağıtımı yeteneğine eklenmesi sağlanarak, "**derleme zamanında "yayımcı-abone" kalıbı servisleri paylaşımı**" gerçekleştirilmiştir.

Bununla beraber, veri sınıfının verileri bir bütün halinde tutuyor olmasının, yeniden kullanılabilirlik açısından elverişli olmadığı görülmüştür. Verilerin gruplanarak, daha küçük parçalara bölünmesi ve ihtiyaç duyulan veri gruplarını projeye dahil edilmesi, yeniden kullanılabilirliği artırıcı bir yöntem olarak benimsenmiştir.

Ayrıca, yazılım içi entegrasyona yönelik olarak tanımlanan altyapılarının tümünün, yazılım dışı bileşenlerle haberleşmeyi sağlayacak şekilde genişletilmesi mümkündür. İhtiyaç duyulan projelerde, mevcut yapılara, CORBA arayüzleri eklenerek, kolaylıkla sorun çözümlenebilmiştir.

Kaynakça

1. Wikipedia, "Publish-Subscribe"
2. www.omg.org
3. www.corba.org
4. Kahraman G., Kahraman E., Ceylan S. "Silah Sistemlerinde Kullanılan Gömülü Yazılım Mimarileri ve ASELSAN'da Gerçekleştirilen Projelerde Edinilen Tecrübeler", UYMK 2006
5. Ünal V., Kahraman E. "Gerçek Zamanlı Gömülü Sistem Tasarımında ASELSAN Yaklaşımı", UYMS 2007
6. www.uml.org
7. Koray T. "Yazılım Mimarileri Üzerinde "Eclipse" Etkileri: Komuta Kontrol Sistemleri Örnek Analizi, UYMK 2006

Bir Veri Soyutlama Katmanı Uygulaması

Serdar Severcan[1], Abdullah Fişne[2], Ceyhun Özgün[3]

[1,2,3] C/S Enformasyon Sistemleri Ltd. Şti. Teknokent ODTÜ/Ankara
[1] serdar.severcan@cs.com.tr, [2] abdullah.fisne@cs.com.tr, [3] ceyhun.ozgun@cs.com.tr

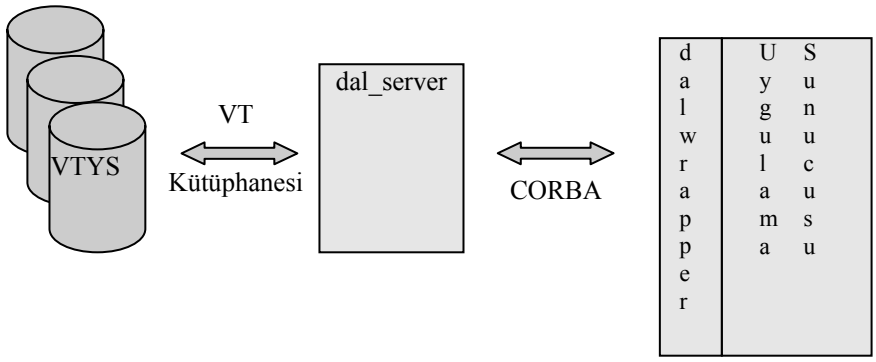
Özet. Kurumsal ölçekli, toplam veri büyüklüğü çok fazla olan (birkaç terabyte ve üstü) projelerde aşağıdaki sorunlar gözlemlenmektedir:
Nesne veri yapısının ilişkisel veritabanı yapısıyla karşılıklı dönüşümleri; uygulama yazılımının veritabanı fiziksel konumuna bağımlılığı; uygulama yazılımının kullanılan veritabanı yazılımına bağımlılığı; kaynakların etkin kullanımı ve ölçeklenebilirlik sorunları; veritabanı sistemlerinin bakım ve yönetiminde karşılaşılan zorluklar; servis tabanlı mimaride veritabanı işlemlerinin (transaction) yönetimini uygulama yazılımı içinden yapmanın zorlukları; performans problemleri; sistem yönetiminde esneklik; iş sürekliliğinin sağlanması. Bu sorunlara etkin bir çözüm sağlamak üzere Veri Soyutlama Katmanı (VSK - DAL) yazılımı geliştirilmiş VEDOP projesi kapsamında kullanılmıştır. Bu çalışmada VSK yazılımının anahtar özellikleri, mimari tasarımı ve sağladığı işlevler anlatılmıştır.

1 Giriş

Veri soyutlama katmanı yazılımları, yoğun şekilde veritabanı işlemleri yapan uygulama yazılımlarının veritabanı sistemlerine olan bağımlılıklarını azaltmak amacıyla ortaya çıkmışlardır. Veritabanı ile ilgili saklama, güncelleme ve sorgulama gibi işlevleri standart bir arayüz aracılığıyla uygulama yazılımlarının hizmetine sunarlar.

2 Mimari

VSK yazılımının mimarisi Şekil-1'de gösterilmiştir. VSK bir CORBA[1] sunucusu (*dal_server*) ve uygulama yazılımına arayüz sağlayan bir modülden (*dalwrapper*) oluşmaktadır. Çok katmanlı mimaride[2,3] bir orta katman uygulaması olarak tasarlanmıştır. dal_server *dalwrapper* tarafından gönderilen komutları yorumlayıp SQL ifadelerine dönüştürür. Bu ifadeleri ilgili veritabanı sunucusunda çalıştırarak sonuçlarını *dalwrapper*'a döndürür. *dalwrapper* gelen sonuçları nesneye çevirerek ya da veri kümesi olarak uygulamaya sunar. Nesne-ilişkisel eşleme işlemi *dalwrapper* modülünde gerçekleştirilir. *dal_server* C++, *dalwrapper* Java ile geliştirilmiştir. *dal_server* ile *dalwrapper* arasındaki iletişim CORBA ile sağlanmaktadır. *dal_server* veritabanlarına veritabanı yazılımının bağlantı kütüphaneleri ile bağlanmaktadır. CORBA gerçekleştirimi olarak Orbix veya omniORB kullanılmıştır. Veritabanı yazılımı olarak Sybase ve Oracle desteklenmektedir. MySQL ve DB2 veritabanları için de deneysel çalışmalar yapılmıştır.



Şekil 1. VSK Genel Sistem Mimarisi

3 İşlev

3.1 Nesne-ilişkisel eşleme

Uygulama yazılımının nesne veri yapısının ilişkisel veritabanı yapısıyla karşılıklı eşlemeleri *dalwrapper* modülünde gerçekleştirilmektedir. Uygulamanın her nesnesi bir sanal tablo ile eşlenmekte, uygulamalar veritabanı işlemlerini bu sanal tablolar üzerinden gerçekleştirmektedirler. *dalwrapper* modülünün PersistenceBroker isimli sınıfı yardımıyla uygulama, bir nesnenin bilgisini veritabanına saklama ya da veritabanından getirme işlemini basit birkaç satır kod ile gerçekleştirebilmektedir.

```
PersistenceBroker pb = DALDB.persistenceBroker();
Tahsilat ths = new Tahsilat();
// set ths fields
pb.insert(TransactionBag.currentTransactionBag(), ths);

PersistenceBroker pb = DALDB.persistenceBroker();
Tahsilat template = new Tahsilat();
template.setOid("oid");
Tahsilat ths = pb.getObjectByTemplate(template);
```

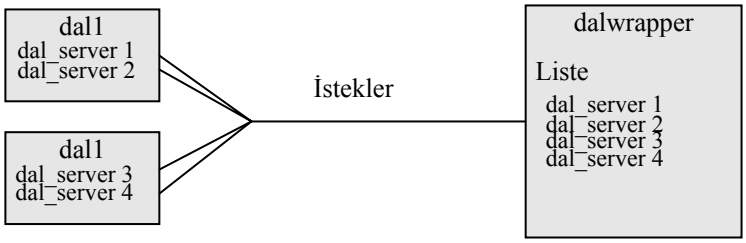
Yukarıdaki örnek kod parçasında `PersistenceBroker` sınıfı yardımıyla bir uygulama nesnesinin veritabanında saklanması ve veritabanından getirilmesi görülmektedir. Her sınıfın sanal tablo ile ne şekilde eşleşeceği bilgisi konfigürasyon dosyalarında belirlenir. Konfigürasyon dosyalarında ayrıca veritabanı sunucuları, sanal tabloların veritabanı tabloları ile eşleşmeleri ve yatay bölümlendirme tanımları yer almaktadır. Konfigürasyon dosyaları *dal_server* tarafından yönetilmektedir. Nesne ilişkisel eşleme için *dalwrapper* tarafından ihtiyaç duyulan konfigürasyon bilgileri, *dalwrapper* ilk yüklenmesi sırasında *dal_server*'dan alınmaktadır.

3.2 İşlem Yönetimi

Yukarıdaki kod parçasında yer alan `TransactionBag` sınıfı *dalwrapper* modülüne ait bir sınıftır. İşlem yönetimini sağlamak amacıyla kullanılmaktadır. Servis tabanlı mimaride servis çağırma zincirindeki bütün servisler bu sınıftan bir nesneyi ortak olarak kullanırlar. Uygulama sunucusunda servis zinciri bir kanal (thread) üzerinde çalışır. Her kanal için bir `TransactionBag` nesnesi hafızada tutulmakta ve servis zinciri için bu nesne kullanılmaktadır. `TransactionBag` nesnesi veritabanı işlemlerinin listesini tutan bir sınıftır. Sorgulama işlemleri içermez. Servis zincirinin sonunda, ilk servisten dönülürken *dalwrapper* veritabanı işlemlerini çalıştırılmak üzere *dal_server*'a gönderir. Bu nesne içerisindeki bütün işlemler *dal_server* tarafından tek bir veritabanı işlemi olarak çalıştırılır. Bu veritabanı işlemi birden çok veritabanı sunucusunda çalışabilecek dağıtık bir işlem olabilir. Dağıtık işlemler için veritabanı yazılımlarının XA standardını[4] desteklemesi gerekmektedir. Dağıtık işlem yönetimi Orbix Object Transaction Service yazılımı yardımıyla yapılmaktadır.

3.3 Yük Dengeleme

VEDOP projesinde *dal_server* sunucusu 3 ayrı makine üzerinde, uygulama yazılımları da 9 adet makinede uygulama sunucusu üzerinde çalışacak şekilde planlanmıştır. Her bi makine 8 adet işlemci içermektedir. Bu ve benzeri kurumsal ölçekli projelerde gelen isteklerin sunucular üzerinde dağıtılmasını sağlayacak bir yük dengeleme mekanizması gereklidir. Başlangıçta *dal_server* için yük dengeleme işlemi Orbix Name Service yazılımı ile yapılmıştır. Bu yazılım CORBA Name Service (isimlendirme servisi) standardına yük dengeleme eklentisi yapan bir yazılımdır. *dal_server* yazılımı başladığında bir isim ve kendisine ait CORBA nesne referansı ile isimlendirme servisine kaydolur. İstemci uygulama (*dalwrapper*) isimlendirme servisinden *dal_server* ismine karşılık gelen nesne referansını ister ve isteklerini bu nesne referansına gönderir. Yapılan yük testleri sonucunda her makinede bir *dal_server* sunucusu çalıştırıp, her istek için isimlendirme servisinden sorgu yapılmasının etkin olmadığı görüldü. Yük testlerinde her işlemci (CPU) için bir adet *dal_server* çalıştırılmasının daha etkin olduğu görüldü. Ayrıca isimlendirme servisinin de her istek için sorgu alması durumunda çok fazla işlemci gücü kullandığı gözlemlendi. Bunun sonucunda *dalwrapper* tarafından bir *dal_server* CORBA nesne referans listesinin tutulması ve isteklerin bu listedeki nesnelere sırayla gönderilmesine karar verildi. Şekil-2'de görüldüğü gibi *dalwrapper* tarafından bütün *dal_server* sunucularının listesi tutulmaktadır. İstekler bu listedeki sunculara round-robin algoritması[5] ile gönderilmektedir. Liste, isimlendirme servisinden *dalwrapper* konfigürasyon dosyasında tanımlanan aralıklarla ya da herhangi bir *dal_server* sunucusuna ulaşılamadığında güncellenmektedir.



Şekil 2. Yük Dengeleme

3.4 Veritabanı Bağımsızlığı

VSK kullanılarak yapılan veritabanı işlemleri *dalwrapper* tarafından sağlanan arayüz ile gerçekleştirilmektedir. Bundan dolayı kullanılan veritabanı yazılımının değiştirilmesi uygulama kodunu etkilemez. Sadece saklı yordamlar kullanılıyorsa bunların yeni veritabanına göre yeniden yazılması gereklidir. Bunlarda veritabanına bağlı özellikler kullanıldıysa eksiklikler olabilir. Bu durum da aslında VSK yazılımının bir eksikliği değildir. Uygulama içinde saklı yordamın kullanımında herhangi bir değişiklik olmamıştır.

3.5 Veritabanı Fiziksel Konum Bağımsızlığı

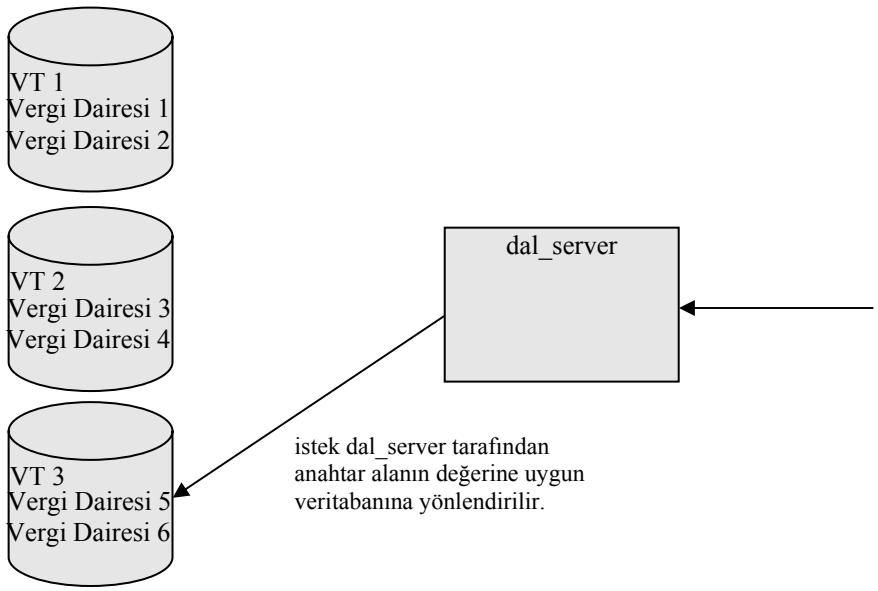
Uygulamalar sanal tablolar üzerinden işlem yaptıkları için veritabanının fiziksel konumundan haberdar değillerdir. Veritabanı fiziksel konumu ile ilgili bilgiler konfigürasyon dosyalarının içerisindedir. Konfigürasyon dosyalarında yapılacak değişikliklerle bir uygulamayı farklı bir veritabanına yönlendirmek mümkündür. Ayrıca aynı veritabanı tablosunun birden çok tanımı yapılarak, farklı uygulamaların farklı yerdeki tabloları kullanması da sağlanabilir. Aynı veritabanı, farklı uygulamalar için farklı isimlerle tanımlanabilir. Böylece veritabanı açısından erişim ve kaynak yönetiminde de esneklik sağlanabilir. Bu yöntemlerin hepsi değişik biçimlerde VEDOP projesinde uygulanmıştır. Rapor uygulama sunucuları isimlendirme servisine farklı isimle kaydedilmiş bir *dal_server* grubuna yönlendirilerek, kopya (replication) veritabanından sorgulama yapmaları sağlanmıştır. Aynı veritabanı farklı uygulamalar için değişik isimlerle tanımlanmış veritabanına bağlantı parametreleri değiştirilmiştir. Sadece sorgu yapan uygulamanın işlem yapmasının önüne geçilmiştir. Veritabanlarının yük durumuna göre sadece sorgu yapan uygulamalar değişik veritabanı sunucularına (kopya-gerçek-2.kopya gibi) yönlendirilebilmiştir.

3.6 Yatay Bölümlendirme

Kurumsal ölçekli projelerde veritabanı boyutu zaman içerisinde çok büyük hale gelebilir. Özellikle çok işlem gören tabloların satır sayısı ve fiziksel boyutu çok büyüyebilir. Çok büyük veritabanlarının bakımı ve yönetiminde çeşitli sorunlar ortaya çıkmaktadır. Tabloya yeni bir kolon eklenmesi, tabloların kopyalarının tutulması (replication), indeks yaratılması, veritabanı istatistiklerinin yenilenmesi gibi işlerin yapılması çok zor ya da çok uzun süreli olmaktadır. Bu veritabanının çalışabilmesi için

çok sayıda işlemciye sahip bir sunucuya ve büyük disk sistemlerinine ihtiyaç duyulmaktadır. Eğer tabloları bir anahtar alana göre yatay olarak bölmek mümkün olursa bütün bu sorunlar çözümlenebilir. Bir veritabanı yerine daha küçük boyutlarda çok sayıda veritabanı kullanılabilir.

VEDOP projesinde karşılaşılan veya karşılaşılması beklenen bütün bu sorunların çözümü için yatay bölümlendirme işlevi VSK tarafından gerçekleştirilmiştir. Örneğin, içerdiği veri miktarı 200 GB'a ulaşmış tablolar vardır. Toplam veri büyüklüğü 3 TB'ı aşmış durumdadır. Bütün vergi daireleri otomasyona geçtikten sonra toplam veri büyüklüğünün 6 TB'ı aşması beklenmektedir. Zaman içerisinde veri büyüklüğü artan veritabanlarında gözlemlediğimiz işlem ve sorgu performasındaki düşüşler, bu büyüklükteki bir veritabanının performans sorunları yaşayabileceğini bize göstermektedir. Bu sebeple bütün vergi dairelerinin verilerinin bulunduğu tek bir veritabanı yerine, Şekil 3'te görüldüğü gibi vergi daireleri gruplarının bulunduğu çok sayıda veritabanı oluşturulmuştur. Bölümlendirme işlemi vergi dairesinin organizasyon koduna göre yapılmıştır. Bölümlendirme ile ilgili, hangi vergi dairesi verisinin hangi veritabanında olduğu bilgisini tutan bir konfigürasyon dosyası *dal_server* tarafından kullanılmaktadır. Bir vergi dairesinin bütün verileri bir bölümde yer almaktadır. Bölümlendirilmiş veri üzerinde yapılan sorgulamalarda sanal tablolardan en az birinin anahtar alanı için, sabit bir değere eşitlik şeklinde kriter verilmelidir. Bu zorunluluk güncelleme ve silme işlemleri için de geçerlidir. Yeni bir kayıt oluşturulması işleminde ise bu alanın verisinin dolu olması ve değerinin geçerli olması gereklidir. Verilen anahtar değer geçerli değilse (konfigürasyon dosyasında verilen değerlerden biri değilse) *dal_server* tarafından bir aykırı durum (exception) oluşturulmaktadır.



Şekil 3. Yatay Bölümlendirme

Olayı veritabanı açısından inceleyecek olursak select, update ve delete ifadelerinde kriterler arasında `anahtar = 'değer'` şeklinde bir kriter olmalıdır. Insert ifadesinde anahtar alan için değer verilmiş olması zorunludur. Select ifadelerinde birden çok bölünmüş tablo varsa, bir tanesi için anahtar değer verilmesi; diğer tablolar için join ifadelerinin olması yeterlidir. Parametre tabloları her veritabanında bulunmaktadır. Parametre tabloları ile bölünmüş tablolar aynı sorgu içinde yer alırlarsa, bölünmüş tablonun bulunduğu veritabanındaki parametre tabloları kullanılmaktadır. Saklı yordamlar da bölümlendirilmiş olarak kullanılabilmektedir. Bölümlendirilmiş tabloların hepsi aynı veritabanı yazılımını kullanmalıdır.

3.7 Diğer İşlevler

Büyük raporların sistem kaynaklarını bitirmeden çalışabilmesini sağlamak amacıyla parçalı veri çekme mekanizması geliştirilmiştir. Sorguya ait sonuç verisinin tamamı bir seferde değil istenilen büyüklükte parçalar halinde uygulamaya verilmektedir. Bu işlem uygulama yazılımının kodunda tek bir değişiklik ile gerçekleştirilmektedir. Normalde bir rapor aşağıdaki kod parçasında gösterildiği şekilde çekilmektedir.

```
Selector select = DALDB.selector();
select.from("vtahsilat");
DALResult result = select.execute();
while(result.hasNext()) {
      result.next();
      // process results
}
```

Parçalı çekme özelliğini kullanmak için uygulamada yapılması gereken tek değişiklik `select.execute()` ifadesinin `select.execute(1000)` ifadesine çevrilmesi olacaktır. İşleyiş uygulamadan bağımsız olarak gerçekleşir. *dalwrapper* istenilen büyüklükteki veri işlendikçe yeniden *dal_server*'dan veri istemektedir.

Ayrıca hatalı sorguların sistem kaynaklarını tüketmesini engellemek için konfigürasyon dosyasından ayarlanabilecek şekilde, tek parça halinde çekilen raporlardan dönebilecek en fazla satır sayısı sınırlanabilmektedir.

Çalışma zamanında değiştirilebilen kademeli bir loglama mekanizması vardır.

Gelen isteklere ait veri ve süreler de tutulmaktadır. Uzun süren isteklere ait bilgiler log dosyalarına yazılmaktadır. İsteklere ilişkin verinin tutulup tutulmayacağı çalışma zamanında değiştirilebilir bir özelliktir.

4 Sonuç

Bu çalışmada çok katmanlı mimariye sahip kurumsal ölçekli projelerde veri soyutlama katmanı incelenmiştir. Bu kısımda kullanım sürecinde sağlanan faydalar ve zorluklar aktarılmıştır.

Farklı uygulamalar (e-beyanname, e-VDO, e-Tahsilat gibi) için farklı isimlerle *dal_server* sunucuları isimlendirme servisine kaydedilmiştir. Sistem yönetimi ve

ölçeklenebilirlik açısından bu yapı önemli bir esneklik sağlamıştır. Bir uygulamanın yükünün daha fazla olduğu bir durumda bu uygulama için fazla sayıda *dal_server* sunucusu açılabilmektedir. Böylece sistemin yük altında kırılmamasının yanısıra, belirli bir servis süresinde cevap verebilmesi de sağlanabilmektedir. Farklı uygulamalar farklı veritabanı kopyalarına yönlendirilerek veritabanları açısından daha iyi bir kaynak yönetimi gerçekleştirilmektedir. Bakıma alınması gereken bir veritabanı sunucusu yerine sistem kolaylıkla yedeğine yönlendirilerek iş sürekliliği sağlanmaktadır.

Bu faydalara karşılık zorluklar da olmuştur. Sistemin yönetim ve gözlemlenmesi ihtiyaçları artmıştır. Bu ihtiyaçları karşılamak amacıyla çeşitli kabuk betikleri (shell scripts) yazılmıştır. Sistemi bir bütün olarak ele aldığımızda çok fazla parametreye sahip olduğunu görmekteyiz. Bu ise sistemin toplam performansının eniyilenmesini zorlaştırmaktadır.

Kaynakça

1. Object Management Group Common Object Request Broker Architecture (CORBA/IIOP) Specification http://www.omg.org/technology/documents/corba_spec_catalog.htm
2. http://n-tier.com/articles/csovervw.html
3. http://www.dossier-andreas.net/software_architecture/tiers.html
4. Distributed Transaction Processing: The XA Specification ISBN: 1 872630 24 3
5. http://en.wikipedia.org/wiki/Round-robin_scheduling

Küçük Mobil Cihazlarda Kablosuz Ağlar Üzerinde SOAP, RMI ve TCP Performans Analizi[1]

Halûk Gümüşkaya[1], Ahmet Volkan Gürel[2], Mustafa Veysi Nural[3]

[1,2,3] Fatih Üniversitesi, Bilgisayar Mühendisliği Bölümü,
34500, Büyükçekmece, İstanbul
[1] haluk@fatih.edu.tr, [2] avgurel@fatih.edu.tr, [3] mnural@st.fatih.edu.tr

Özet. Bu çalışmada ortamdan haberdar (context-aware) bir sisteme, Bluetooth ve IEEE 802.11 gibi kablosuz ağlar üzerinden küçük mobil cihazlar ile, kısa mesafeli erişimlerde kullanılabilecek ağ mimarileri incelenmekte ve analiz edilmektedir. Bu analizlerde, dağıtık sistemlerde üç önemli neslin temsilcisi, TCP soket bağlantısı, RMI dağıtık nesne teknolojisi ve servis odaklı yaklaşımlar kullanılarak, istemci/sunucu mimarileri incelenmekte ve değişik kablosuz ağ bağlantılarındaki ağ paket ölçümleri yapılmakta, mimari performans analizleri sunulmaktadır. Bu analizlerin sonucunda, günümüzde mevcut mobil cihazlara yönelik, değişik uygulamaların ihtiyaçlarına göre seçilebilecek yazılım mimarileri ve teknolojileri değerlendirilmektedir.

1 Giriş

Bluetooth ve Wi-Fi (IEEE 802.11) gibi kısa mesafe kablosuz erişim teknolojilerinde, 2000'li yılların başından günümüze çok hızlı gelişmeler olmaktadır. Önce bilgisayarlara ve küçük mobil cihazlara dışarıdan takılan kart desteğiyle gelen bu kablosuz ağ bağlantıları, artık cihazların üzerinde gelmektedir. Bu teknolojilerin hem kapsama alanları genişlemiş hem de bağlantı hızları artmıştır. Bluetooth teknolojisi hemen hemen bütün cep telefonlarının ve bilgisayarların bütünleşik bir parçası olmuştur. Son üç dört yıldır masaüstü ve taşınabilir bilgisayarların üzerinde gelen IEEE 802.11 teknolojisi, artık yavaş yavaş cep telefonlarına da entegre edilmektedir.

Bluetooth ve IEEE 802.11 gibi kablosuz kısa mesafe bağlantı teknolojilerine sahip küçük mobil cihazlar için uygulamalar geliştirme yazılım mühendisliğinin önemli bir araştırma konusu olmuştur [1], [2]. Bu çalışmada cep bilgisayarları ve cep telefonlarında *çalışabilen*, sistemden bağımsız *taşınabilir*, yazılım teknolojileri ve mimarileri incelenmekte ve performans analizleri sunulmaktadır. Bu analizlerde temel TCP (Transmission Control Protocol) soket bağlantısı, RMI (Remote Method Invocation) dağıtık nesne teknolojisi ve servis odaklı yaklaşım kullanılarak, istemci/sunucu bağlantı mimarileri incelenmekte ve değişik kablosuz ağ bağlantılarındaki zaman analizleri sunulmaktadır.

[1] Bu çalışma P50050702 proje numarası ve *Frameworks for Context-Aware Pervasive Systems─FCAPSYS* adı ile Fatih Üniversitesi Bilimsel Araştırma Fonu tarafından desteklenmektedir.

2 Önceki Çalışmalar

Mobil cihazlar ve dağıtık sistemlere yönelik varolan ve yeni çıkan standartların kullanıldığı *Service-Oriented Wireless Context-Aware System (SOWCAS)* adında *ortamdan haberdar (context-aware)* bir sistem (framework) geliştirmekteyiz [3], [4], [5]. SOWCAS'daki temel amacımız, bir üniversite kampüsündeki kablosuz erişim teknolojilerinin heterojenliğinden yararlanan, bina içinde ve dışında çalışan ortamdan haberdar bir sistem geliştirmektir. Genel olarak servis odaklı yaklaşımla geliştirilen bu sistemdeki mobil istemcilerin, kısa mesafeli kablosuz bağlantılar için değişik yazılım teknolojilerinin ve mimarilerinin incelenmesi ve performans analizlerinin yapılması gerekiyordu.

Servis odaklı yazılım geliştirme üzerine çalışmaların yaygınlaşmaya başlamasıyla kablolu ağlar için, web servisleri ve SOAP ile RMI ve CORBA gibi ortakatman (middleware) dağıtık nesne teknolojilerinin performans analizini karşılaştıran birçok çalışma yapılmıştır [6], [7], [8], [9], [10], [11]. Bu çalışmalarda, genellikle basit bir metodu çağıran bir web servis uygulaması ile aynı metodu çağıran RMI, CORBA veya diğer gerçekleştirmenin performansı karşılaştırılmıştır. Daha önceki bu çalışmalar, mevcut web servis çözümlerinin RMI/CORBA uygulamalarına göre önemli ölçüde daha kötü performansa sahip olduğunu göstermiştir. İlk çalışmaların sonuçları 400:1 gibi büyük performans değerleri göstermiştir.

Bizim çalışmamızın daha önceki çalışmalardan önemli farkları bulunmaktadır. Birinci fark, günümüzdeki küçük mobil cihazlar için bu performans testlerinin yapılmış olmasıdır. Cep telefonu ve cep bilgisayarı gibi mobil cihazların en yaygın özellikleri arasında, sınırlı kaynaklar (mikroişlemci, hafıza, giriş/çıkış yetenekleri gibi), heterojenlik ve yüksek düzeyde dinamizm gelmektedir. Kaynakların sınırlılık durumları, cihazdan cihaza değişkenlik gösterse de, mobil cihazlar masaüstü bilgisayarlar kadar güçlü işlemcilere, büyük hafızalara, yüksek hızlı G/Ç arabirimlerine ve ağ yeteneklerine sahip değildir. RMI ve CORBA gibi geleneksel ortakatman platformları ve SOAP gibi yeni teknolojiler bazı mobil cihazlar ve kablosuz uygulamalar için uygun olmayabilir.

Performans çalışmamızın ikinci önemli farkı, testlerin değişik kablosuz bağlantılar üzerinde yapılmış olmasıdır. Daha önceki performans çalışmalarının hepsi kablolu ağlar üzerinde sabit güçlü bilgisayarlarda gerçekleştirilmiştir.

Çalışmamızın üçüncü ve son önemli farkı, programların taşınabilir olmasını istediğimizden dolayı mobil cihazlarda Java'nın kullanılmasıdır. Java'nın çalıştığı ortamlardaki JVM sanal makinesinin getirdiği yavaşlık ve Java'nın küçük mobil cihazlarda çalışması önündeki diğer sınırlamalar, performans çalışmamızın ayrıntılarını ve zorluklarını oluşturmuştur.

Son zamanlarda web servisler için yeni nesil hızlı web servis geliştirme yazılım ortamları ortaya çıkmış olmasına rağmen, cep bilgisayarları ve diğer mobil cihazlar için oldukça az sayıda servis odaklı Java ortamı bulunmaktadır [12]. Performans testlerinde dağıtık nesne teknolojilerinin temsilcisi olarak seçmiş olduğumuz RMI için de durum

aynıdır. Küçük mobil cihazlar için RMI paketleri de oldukça az sayıdadır ve masaüstü bilgisayarlarda olduğu kadar güçlü değildir.

3 Küçük Mobil Cihazlar İçin Ağ Bağlantıları

3.1 Kablosuz Kısa Mesafeli Erişim Teknolojileri

Bluetooth, taşınabilir ve sabit elektronik cihazlar için kablo bağlantısını ortadan kaldırmak amacıyla tasarlanan, kısa menzilli ve düşük maliyetli bir kısa mesafe kablosuz bağlantı teknolojisidir [13], [14]. 2.4 GHz ISM frekans bandında çalışan ve düşük güç tüketimine sahip olan Bluetooth teknolojisi ile teorik olarak maksimum 100 metre mesafede, 1Mbit/s (versiyon 1.2) - 3Mbit/s (versiyon 2.0+EDR) hızında kablosuz kişisel alan ağları (Wireless Personal Area Network - WPAN) oluşturulabilir.

Wi-Fi (Wireless Fidelity) (IEEE 802.11) bilgiye kolay erişim, kablolamadan kurtulma ve uyumluluğun sağlanması hedefleriyle ortaya çıkan, genellikle taşınabilir elektronik cihazlar için Bluetooth'a göre daha yüksek hız ve uzun mesafeli erişim sağlayan, bir kısa mesafe kablosuz bağlantı teknolojisidir [15], [16]. Özellikleri IEEE 802.11 standartlarına göre belirlenir ve şu anki cihazlarda yaygın olarak a/b/g/n standardları mevcuttur. Desteklenen standarda göre 2.4 veya 5 GHz ISM frekans bandında çalışan ve Bluetooth'a göre daha yüksek güç tüketime sahip olan Wi-Fi ile teorik olarak maksimum 250 metre mesafede, teorik limitler olarak 11 Mbit/s (802.11b), 54 Mbit/s (802.11a/g) ve 300 Mbit/s (802.11n) hızlarında kablosuz yerel alan ağları (Wireless Local Area Network - WLAN) oluşturulabilir.

Günümüzde satılan küçük taşınabilir cihazların tamamında Bluetooth teknolojisi bulunmaktadır, ancak daha hızlı veri aktarımını sağlayan versiyon 2.0 ve EDR (Enhanced Data Rate) desteği çok azında sunulmaktadır. Yine Wi-Fi desteği küçük cihazların çoğunda bulunmakla birlikte, hızlı ve uzun menzilli bağlantıyı sağlayan 802.11g ve 802.11n desteği çok az cihazda sunulmaktadır.

3.2 Kablosuz Kısa Mesafeli Bağlantılar İçin Yazılım Teknolojileri

Diğer çalışmalarımızda sunulan [3], [4], [5] SOWCAS mimarisi geliştirilirken, küçük mobil cihazlar ile yapılan kısa mesafeli kablosuz bağlantılardaki, temel TCP/UDP soket bağlantıları, RMI dağıtık nesne teknolojisi ve servis odaklı yaklaşım kullanılarak istemci/sunucu mimarileri incelendi. Geliştirilen test programları kullanılarak bir istemci ile sunucu arasındaki değişik kablosuz ağ bağlantılarının paket ölçümleri ve zaman analizleri detaylı olarak yapıldı.

UDP bağlantısız ve güvenilir olmayan bir aktarım katmanı protokolüdür ve genellikle paket kayıplarına toleranslı ve hıza duyarlı, ses ve görüntü gibi, akan çoklu ortam (streaming multimedia) uygulamalarında kullanılmaktadır. Çalışmamızda RTP (Real Time Protocol) ve RTSP (Real Time Streaming Protocol) uygulama protokollerini kullanarak istemcilerine video klipler dağıtan UDP tabanlı küçük bir Java çoklu ortam sunucusu geliştirdik. Herhangi bir özel kütüphane kullanmadık. Programlarımız paketleme işlemini RTP ve sunucu/istemci etkileşimini RTSP kullanarak ele almaktadır. Java RTSP istemci uygulamasının 416 MHz PDA üzerindeki performansı kabul edilebilir düzeydeydi. Bir sonraki bölümde bu sonuçlar verilmemektedir. Çünkü RMI

ve SOAP teknolojilerinin aktarım katmanları TCP protokolünü kullandığı için, RMI ve SOAP uygulamalarının karşılaştırılmasında TCP protokolü referans olarak kullanıldı. Mobil cihazlar için UDP test sonuçları, gerçek zamanlı çoklu ortam uygulamalarının anlaşılmasında çok faydalı bir görüş sağladı.

Servis Odaklı Bilgi İşleme (SOBİ), servisleri kullanan yazılım uygulamalarının geliştirilmesi için mimari model olan Servis Odaklı Mimari (SOM) tabanlı bir dağıtık bilgi işleme yaklaşımıdır [17]. SOBİ ve SOM tamamen yeni kavramlar değildir, CORBA ve RMI gibi diğer dağıtık bilgi işleme teknolojileri de benzer kavramlar kullanmaktadır. SOBİ ve SOM varolan kavramların ve XML, SOAP gibi platformdan bağımsız dağıtık sistemlerin gerçekleştirilmesinde kullanılan yeni teknolojilerin devamı niteliğindedir.

SOM mobil servisler için ideal bir yaklaşım olarak görülmektedir [3], ancak şu anda kurumsal ve ticari servisler üzerinde odaklanmıştır. Ayrıca, SOM ile ilgili araştırmaların büyük bölümü kablolu ağlara yönelik mimariler ve uygulamalar üzerinde odaklanmıştır. SOM tabanlı kablosuz ara katman ile ele alınması gereken bir çok problem bulunmaktadır. Kablosuz ara katman, servis odaklı uygulamaların hazırlanması ve yönetilmesinde çok önemli bir rol oynayacaktır. Bu araştırmanın ana hedeflerinden birisi, mobil servislerin gerçekleştirilmesinde SOM yaklaşımından nasıl yararlanılabileceğini incelemektir.

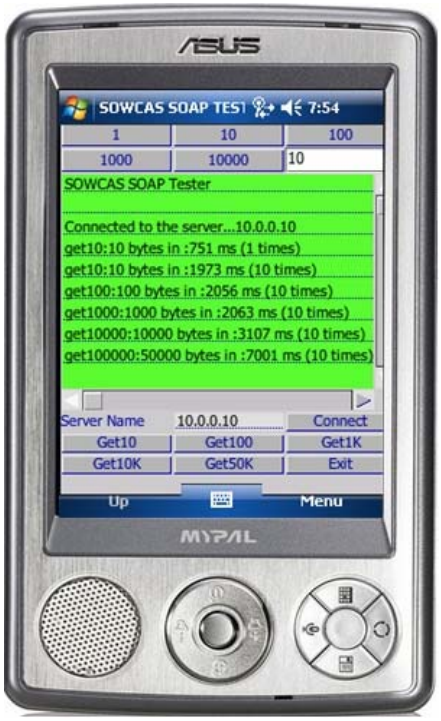
4 Kablosuz Bağlantıların Performans Analizi

4.1 Performans Ölçümü Test Cihazları, Yazılımları ve Test Yöntemleri

Çalışmamızda, küçük mobil cihazlarda çalışan TCP soketleri, RMI ve SOM tabanlı kablosuz uygulamaların performansının kabul edilebilir olup olmadığını görmek amacıyla, kablosuz bağlanan bir mobil istemcinin performansını ölçen karşılaştırmalı testler gerçekleştirdik. Aşağıda önce test cihazlarımız ve kullandığımız yazılımlar sunulacak, daha sonra testlerimiz verilecektir.

Performans testlerinde sabit bir sunucu ve mobil elde-tutulan bir istemci cihazı kullanıldı. Test sunucusu olarak Windows XP SP2 üzerinde çalışan, Intel Centrino 1.86 GHz işlemci ve 1GB RAM içeren bir diz üstü bilgisayar kullanıldı. Ağ trafik analizi, diz üstü sunucu ile mobil istemci arasındaki 11/54 Mbit/s WLAN bağlantı ve 700 Kbit/s (Bluetooth 1.2) / 3 Mbit/s (Bluetooth 2.0+EDR) WPAN kablosuz hatlar üzerinden aktarılan paketlerin yakalanması için Wireshark ağ paket analiz aracı [18] ile yapıldı. Sunucu tarafında kullanılan yazılım teknolojileri Jakarta Tomcat 5.5, Axis 1.3 ve Java 1.5 SDK'dır.

Mobil istemci ise Windows Mobile 6.0 ile çalışan, 416MHz Intel XScale işlemcili, 128 MByte Flash ROM, 64 MB SDRAM, IEEE 802.11 b+g, GPS ve Bluetooth 2.0+EDR özelliklerine sahip Asus MyPAL A636N cep bilgisayarıdır. İstemci tarafında kullanılan yazılım teknolojileri kSOAP kütüphanesi [19] ve CrEme JVM 4.11'dir [20]. Testlerimizdeki mobil cihaz için geliştirdiğimiz, TCP, RMI ve SOAP ile yapılan kablosuz ağ bağlantılarının hepsinde aynı kullanıcı arayüzüne sahip Java istemci test programı Şekil 1'de gösterilmektedir.



Şekil 1. Performans test programı kullanıcı arayüzü

Performans testleri için kullanılan uzak arabirimin sadeleştirilmiş hali aşağıda verilmiştir

```
public interface SOWCASTest extends Remote {
  public String get10Bytes() throws RemoteException();
  public String get100Bytes() throws RemoteException();
  public String get1KBytes() throws RemoteException();
  public String get10KBytes() throws RemoteException();
  public String get50KBytes() throws RemoteException();
}
```

Testlerimizde istemci tarafındaki hafıza kullanımı, istemci ve sunucu tarafında CPU kullanımı, tek bir istek için ve bir dizi istek için toplam gecikme süreleri, aktarılan toplam byte miktarı ve paket sayıları ölçülmüştür. İlk testlerde küçük (10 byte), orta (100 ve 1000 byte) ve büyük (10.000 ve 50.000 byte) veri setleri ile çok sayıda TCP, RMI ve SOAP erişimi yapılarak, her test için istemci tarafındaki ortalama CPU süreleri hesaplanmıştır.

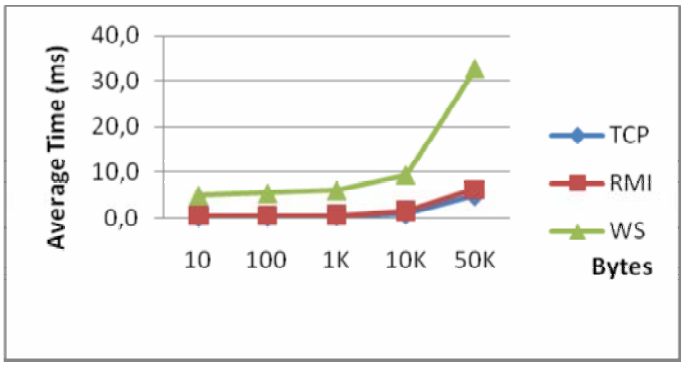
4.2 Kablosuz Ağlarda Paket Ölçümleri ve Performans Analizi

Daha sonra yapacağımız kablosuz ağlardaki testlerimizi karşılaştırmak için bir referans olsun diye, ilk testlerimizi önce hızlı kablolu 100 Mbit Ethernet ağında gerçekleştirdik. Bulmuş olduğumuz sonuçlar genel olarak daha önceki araştırmacıların sonuçlarını doğrulamakta; bununla birlikte yeni daha hızlı SOAP yazılımlarından dolayı çok daha iyi sonuçlar aldık. Kablolu ağ ortamı için bulmuş olduğumuz istemci makinesinde CPU ortalama test sonuçları Tablo 1'de ve karşılaştırdığımız teknolojilerin birbirine göre performans oranları da Şekil 2'de verilmiştir.

Tablo 1. TCP, Web Servis (WS) ve RMI test programlarının kablolu 100 Mbit Ethernet ağındaki ortalama CPU süreleri ve performans oranları

	İstemci Ortalama CPU Zamanları (ms)				
# byte'lar	**10**	**100**	**1K**	**10K**	**50K**
TCP	0,2	0,3	0,4	0,9	4,8
RMI	0,5	0,6	0,7	1,5	6,2
WS	4,8	5,3	6,0	9,4	32,9
	Performans Oranları				
WS/TCP	28,5	21,3	14,9	10,6	6,8
RMI/TCP	3,2	2,4	1,7	1,8	1,3
WS/RMI	9,0	9,0	9,0	6,1	5,3

K = 1000 byte



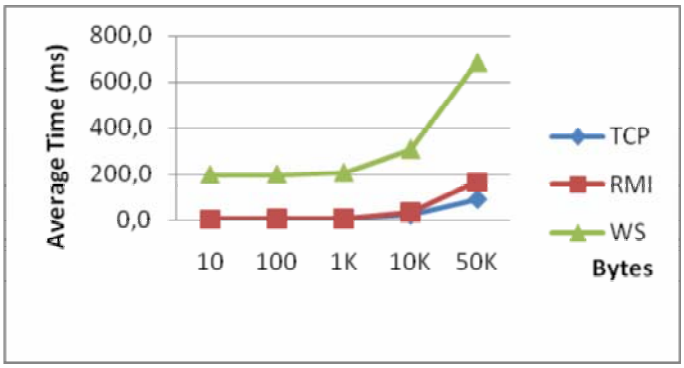
Şekil 2. TCP, RMI ve Web Servis test programlarının 100 Mbit Ethernet ağındaki performans oranları

SOWCAS'ta istemci bağlantılarında genellikle IEEE 802.11 kablosuz bağlantısı kullanılacağı için, daha çok bu bağlantıların değişik mesafelerden performans analizleri yapıldı. Aşağıdaki Tablo 2, Şekil 3 ve Şekil 4'te bu sonuçlar sunulmaktadır.

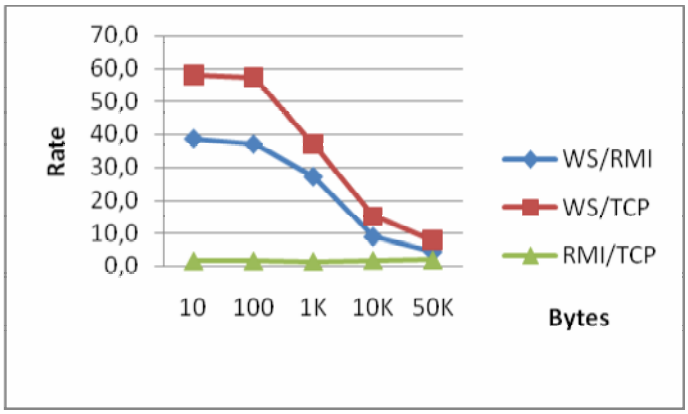
Tablo 2. İstemci tarafında TCP, Web Servis (WS) ve RMI test programlarının kablosuz 54 Mbit IEEE 802.11 bağlantısındaki ortalama CPU süreleri ve performans oranları

	İstemci Ortalama CPU Zamanları (ms)				
# byte'lar	**10**	**100**	**1K**	**10K**	**50K**
TCP	3,4	3,5	5,6	20,7	90,0
RMI	5,1	5,4	7,6	35,0	164,7
WS	197,0	199,4	206,4	308,3	685,6
	Performans Oranları				
WS/TCP	57,9	57,0	36,9	14,9	7,6
RMI/TCP	1,5	1,5	1,4	1,7	1,8
WS/RMI	38,6	36,9	27,2	8,8	4,2

K = 1000 bytes



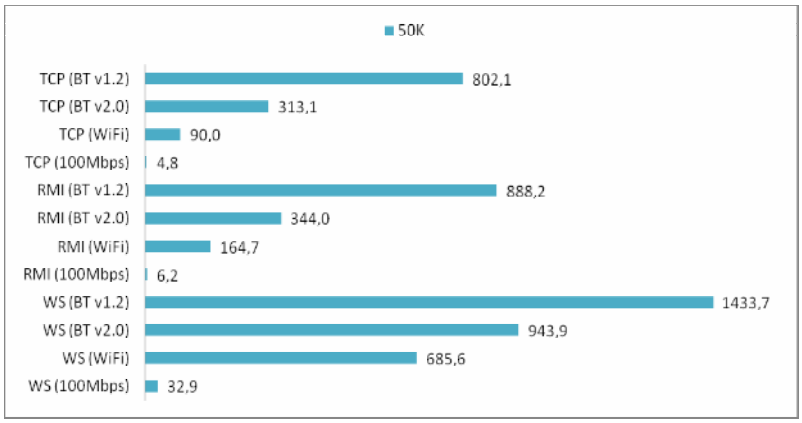
Şekil 3. TCP, RMI ve Web Servis test programlarının 54 Mbit WLAN bağlantısındaki performansları



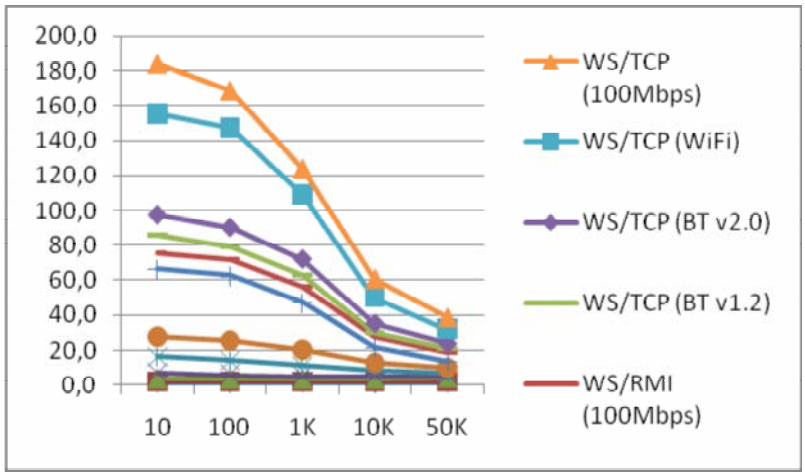
Şekil 4. TCP, RMI ve Web Servis test programlarının 54 Mbit WLAN bağlantısındaki performans oranları

IEEE 802.11 ile başlamış olduğumuz kısa mesafeli kablosuz bağlantıların performans testlerinde, daha sonra, çok daha kısa mesafe (1, 3 ve 6 metre) testlerinde Bluetooth 1.2 (700 Kbit bağlantı hızı) ve Bluetooth 2.0+EDR (3 Mbit bağlantı hızı ve Bluetooth 1.2'ye göre daha uzak mesafeden kablosuz bağlantı) ile bağlantılar gerçekleştirdik.

Aşağıdaki Şekil 5'te mobil cihaz ile sunucu arasındaki, Wi-Fi (54 Mbit), Bluetooth 1.2 ve Bluetooth 2.0+EDR kablosuz bağlantılardaki, 50 Kbyte veri aktarımının TCP, RMI ve Web Servis teknolojileri kullanılarak bulunan performans sonuçları verilmektedir. Şekil 6'da ise, bütün testlerimizin karşılaştırılmalı performans oranları verilmektedir.



Şekil 5. TCP, RMI ve Web Servis test programlarının kablolu (100 Mbit Ethernet) ve kablosuz hatlardaki (Wi-Fi (54 Mbit), BT 2.0+RDR (3 Mbit), BT 1.2 (700 Kbit)) performansları (ms)



Şekil 6. TCP, RMI ve Web Servis test programlarının kablolu (100 Mbit Ethernet) ve kablosuz hatlardaki (Wi-Fi (54 Mbit), BT 2.0+EDR (3 Mbit), BT 1.2 (700 Kbit)) performans oranları

5 Sonuçlar

Yukarıda verilen test sonuçlarımıza göre, web servis uygulamaları RMI ve TCP soket uygulamalarına göre daha yavaş çalışmaktadır. RMI uygulamaları küçük miktardaki dokümanlar ve düşük gecikmeli ağlar için daha hızlı çalışmaktadır, ancak büyük miktarda dokümanlar ve yüksek gecikme olduğunda performans hızla düşmektedir. Web servisler yüksek başlangıç maliyetine sahip olmakla birlikte, veri miktarı büyüdükçe maliyet ya çok az değişmekte ya da değişmemektedir. Yüksek gecikme çok daha büyük başlangıç maliyetleri getirse de performans veri büyüklüğünden bağımsızdır. Ağ gecikmesi arttıkça, testlerimizde 100 Mbit Ethernet bağlantısından 54 Mbit Wi-Fi bağlantısına, oradan da 3 Mbit Bluetooth 2.0+EDR ve 700 Kbit Bluetooth 1.2 bağlantılarına düşüldükçe, doküman odaklı yaklaşımın performans getirileri de önemli ölçüde artmaktadır. Bağlantıların sıkça koptuğu veya asenkron işlemlerde gecikmenin dakikalarla ölçüldüğü bazı gerçek dünya kablosuz haberleşme senaryolarında bu durum çok önem kazanmaktadır.

Performans çalışmaları genellikle RPC tarzı haberleşmeyi ölçmektedir ve web servislerin asıl gücünü gösterecek olan doküman odaklı tasarımların sağladıklarını göz önüne almamaktadır. Web servisler diğer dağıtık nesne teknolojileri gibi RPC tarzı kullanım için tasarlanmamıştır. Web servisler RPC merkezliden çok, doküman merkezli bir haberleşme modelinde kullanılabilen bir tanımlı kodlama (literal encoding) sağlar. Bir RPC odaklı yaklaşımla karşılaştırıldığında, eğer web servislerin doküman merkezli yapısı uygulamalarda kullanılırsa, diğer geleneksel uygulamalara göre daha iyi sonuçlar alınacaktır.

Web servislerin RMI'ya göre daha fazla ağ bant genişliği ve CPU süresine ihtiyaç duymasının temel sebebi, SOAP mesajlarında bulunan büyük miktardaki XML üstverisidir (metadata). Web servislerde sayılar ve diğer tüm veriler metine (text) çevrilir. Yapıyı tanımlayan üstveri XML biçim imleri (mark-up tag) şeklinde oluşturulur. XML ayrıştırıcıları (parser), istemci ve sunucu uygulamalarının, veri yapıları için kendi ayrık ama birbirine eşit gösterimini oluşturmalarına izin verir. HTTP ve XML metin dokümanlarının kullanımı sayesinde karşılıklı uyumluluk (interoperability) artmaktadır. Bununla birlikte web servis çözümlerinin yürütme zamanı (run-time) maliyeti de RMI çözümlerine göre önemli ölçüde yüksek olur. XML formatlı dokümanlar, diğer yaklaşımlardaki ikili (binary) veri trafiğine göre doğal olarak daha büyüktür. Ağ üzerinde daha fazla veri aktarımı yapmak gerekir ve daha fazla kontrol paketine ihtiyaç vardır.

Sadece performans yönü ele alınırsa, XML ve SOAP ayrıştırmanın (parsing) getirdiği yük, sunucu üzerinde çalıştırılan iş mantığı veya hesaplamadan daha hafif olduğunda web servisler değer kazanmaktadır. Web servisler genellikle iyi performans üretememelerine rağmen kullanıcı arayüzü, otomatik güvenlik duvarı koruması (aktarım için HTTP kullanır ve HTTP genellikle güvenlik duvarlarından geçmektedir), uygulamada taşınabilirlik ve heterojenlik, şeffaf vekiller ve ince (thin) istemciler sağlamada uygundur. RMI'da olduğu gibi, dağıtık nesneler için en doğal tasarımların kullanımı kolay ama ölçeklenebilirliği zayıftır. Web servisler ise iyi ölçekleme özelliklerine sahiptir.

Kaynakça

1. The Pittsburgh Pebbles PDA Project: http://www.pebbles.hcii.cmu.edu.
2. Myers, B. A. "Using Hand-Held Devices and PCs Together", Communications of the ACM. Cilt 44, Sayı 11, Kasım 2001, s. 34 - 41.
3. Yurday, B., Gümüşkaya, H., "A Service Oriented Reflective Wireless Middleware", Lecture Notes in Computer Science (LNCS), Springer-Verlag, Çilt 4294, 2006, s. 545-556.
4. Gümüşkaya, H., Nural, M. V., "Service-Oriented Context-Awareness and Context-Aware Services", Advances in Computer and Information Sciences and Engineering, Springer, Ağustos, 2008.
5. Gümüşkaya, H. Gürel, A. V., Nural, M. V., "Ortamdan Haberdar Bir Sistem İçin Mobil Ağ Yazılım Mimarileri", 2. Ulusal Yazılım Mimarisi Konferansı (UYMK'08), 11-12 Eylül 2008, Ege Üniversitesi, İzmir.
6. Davis, D., Parashar, M., "Latency Performance of SOAP Implementations", IEEE/ACM International Symposium on Cluster Computing and the Grid, May 2002, s. 407-412.
7. Elfwing, R., Paulsson, U., Lundberg, L., "Performance of SOAP in Web Service Environment Compared to CORBA", APSEC, IEEE Computer Society, 2002, s. 84–93.
8. Demarey, C., Harbonnier, G., Rouvoy, R., Merle, P., "Benchmarking the Round-Trip Latency of Various Java-Based Middleware Platforms", Studia Informatica Universalis, Mayıs 2005, s. 7–24.
9. Gray, N. A. B., "Comparison of Web Services, Java-RMI, and CORBA Service Implementations", Fifth Australasian Workshop on Software and System Architectures, Melbourne, Australia, April, 2004.
10. Juric, M. B., Rozman, I., Brumen, B., Colnaric, M., Hericko M., "Comparison of Performance of Web Services, WS-Security, RMI, and RMI–SSL", The Journal of Systems and Software, 2006, s. 689-700.
11. Cook, W. R., Barfield, J., "Web Services versus Distributed Objects: A Case Study of Performance and Interface Design", Proc. of the IEEE International Conference on Web Services, 2006, s. 419-426.
12. Web Servis paketleri: http://www.soaprpc.com/ws_implementations.html.
13. Bluetooth: http://www.bluetooth.com/Bluetooth/Technology/Basics.htm.
14. Bluetooth: http://en.wikipedia.org/wiki/Bluetooth.
15. Wi-Fi (IEEE 802.11): http://standards.ieee.org/getieee802/802.11.html.
16. Wi-Fi (IEEE 802.11): http://en.wikipedia.org/wiki/Wi-Fi.
17. Erl, T., Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall, 2005.
18. Wireshark web site: http://www.wireshark.org.
19. kSOAP: http://sourceforge.net/projects/ksoap2.
20. CrEme web site: http://www.nsicom.com.

YAZARLAR DİZİNİ