3. Ulusal Yazılım Mimarisi Konferansı '10

4-5 Kasım 2010
Bilkent Üniversitesi
Ankara, Türkiye

Bildiriler Kitabı

Editörler

Bedir Tekinerdoğan, Bilkent Üniversitesi

Semih Çetin, Cybersoft

IEEE TURKEY SECTION

Bilkent University

Cybersoft
vision@work to make IT happen

3. ULUSAL YAZILIM MİMARİSİ KONFERANSI '10

4-5 Kasım 2010
Bilkent Üniversitesi
Ankara, Türkiye

Bildiriler Kitabı

Editörler

Bedir Tekinerdoğan, Bilkent Üniversitesi

Semih Çetin, Cybersoft

IEEE TURKEY SECTION

Bilkent University

Cybersoft
vision@work to make IT happen

İÇİNDEKİLER

Gömülü Sistemler

Yazılım Ürün Aile Mühendisliği

ÖNSÖZ

Türkiye'de ilk Yazılım Mimarisi Çalıştayı 2005 yılında Ulusal Yazılım Mühendisliği Semineri kapsamında organize edildikten sonra, 2006 yılında başlamak üzere iki yılda bir Ulusal Yazılım Mimarisi Konferansını (UYMK) organize ediyoruz. 1. UYMK İstanbul'da ve 2. UYMK İzmir'de organize edildikten sonra 3. UYMK'yı Ankara, Bilkent Üniversitesinde organize ettik. UYMK yalnızca yazılım mimarisi ve ilgili konuları içermektedir ve Türkiye'de yazılım mühendisliği konusu ile ilgili pratisyenleri ve akademisyenleri, her yönüyle yazılım mimarisi ile ilgili deneyim, sonuç ve fikir paylaşımı yapmak üzere bir araya getirmeyi amaçlamaktadır.

Daha önceleri bu konferansın temel hedefleri olarak bu konu hakkında bilinci geliştirmek, araştırma ve eğitimi özendirmek, ve yazılım mimarisi tasarımının uygulanmasını özendirmek olarak sıralamıştık. Bu hedeflerin halen çok önemli olduğunu düşünmekteyiz. Daha da geniş perspektifte baktığımızda genel olarak yazılım mühendisliği konusunun ülkemizde daha hızlı gelişmesi için sistematik adımlar atılması gerektiğini düşünüyoruz. Üniversitelerde daha fazla yazılım mühendisliğine önem verilmesi, daha fazla yazılım mühendisliği dersleri verilmesi ve akademi ve sanayinin bu konuda aktif işbirliği yapılması gerekiyor. Bu konferansın organizesi atılması gereken adımlardan sadece bir tanesi olarak değerlendiriyoruz.

Konferansın Program Komitesinde hem akademik çevrelerin hem de yazılım sektörü temsilcilerinin yer almasına önem verilmiştir. Konferansa gönderilen bildirilerin 70% Program Komitesinin değerlendirmeleri sonucunda konferansta sunulmak üzere kabul edilmiştir. Kabul edilen bildiriler Yazılım Mimarisi Tasarımı, Model-Güdümlü Yazılım Mimari, Gömülü Sistemler, Yazılım Ürün Aile Mühendisliği, gibi farklı alanları içermektedir. Bildiri Oturumların yani sıra iki panel de organize edilmiştir. İlk günkü panelde yazılım mimarisi tasarımında kalite yönetimi tartışılmıştır. İkinci gün panelinde ise ülkemizde uygulanan yazılım ürün aile mühendisliği tecrübeleri sunulup tartışılmıştır.

Konferans iki davetli konuşmacıların sunumlarına ve görüşlerine yer vermiştir. IT Copenhagen Üniversitesinden katılan Dr. M. Ali Babar, "Agility and Architecture: Why and how they can co-exist? " başlıklı bir sunum vermiştir. ABD Software Engineering İnstitute'den katılan Dr. Paul Clements "Evaluating The Software Architecture Competence of Organizations" başlıklı sunum vermiştir.

Yazılım mimarisinin uluslararası platformda yıllardır giderek önem kazandığını düşünerek bu konferansın da ülkemizde yazılım mühendisliği disiplinin gelişmesine katkıda bulunacağını ümit ediyoruz. Konferansın organizasyonunda emeği geçen tüm Program Komitesi üyelerine, davetli konuşmacılarımıza, Bilkent Üniversitesi yerel organizasyon kadrosuna, ana sponsorumuz Cybersoft'a ve konferansın tüm katılımcılarına teşekkürlerimizi sunarız.

Bedir Tekinerdoğan, Bilkent Üniversitesi
Semih Çetin, Cybersoft

YEREL ORGANİZASYON KOMİTESİ

Bilkent Üniversitesi, Ankara

H. Altay Güvenir, Bilkent Üniversitesi (bölüm bşk.)

Başak Çakar, Bilkent Üniversitesi

Elif Demirli, Bilkent Üniversitesi

Özgür Aydın Tekin, Bilkent Üniversitesi

Bedir Tekinerdoğan, Bilkent Üniversitesi

Buğra Yıldız, Bilkent Üniversitesi

PROGRAM KOMİTESİ

Program Kurulu Eş Başkanları

Bedir Tekinerdoğan, Bilkent Üniversitesi

Semih Çetin, Cybersoft

Program Kurulu

Naci Akkök, Oslo Üniversitesi

Mehmet Akşit, Twente Üniversitesi

Barış Aktemur, Özyeğin Üniversitesi

Levent Alkışlar, Aselsan

İlker Aktıntaş, Cybersoft

Meriç Aykol, TBD İstanbul

Turgay Aytaç, Logo Business Solutions

Fevzi Belli, Paderborn Üniversitesi

Ayşe Başar Bener, Boğaziçi Üniversitesi

Veli Biçer, FZI

Feza Buzluca, İstanbul Teknik Üniversitesi

Çağatay Çatal, TÜBİTAK

Cengiz Çelik, Bilkent Üniversitesi

Turgay Çelik, Milsoft

Semih Çetin, Cybersoft

Serhan Çiçek, Aselsan

Onur Demirörs, Orta Doğu Teknik Üniversitesi

Oğuz Dikenelli, Ege Üniversitesi

Kıvanç Dinçer, TÜBİTAK

Banu Diri, Yıldız Teknik Üniversitesi

Asuman Doğaç, Orta Doğu Teknik Üniversitesi

Ali Doğru, Orta Doğu Teknik Üniversitesi

Uğur Doğrusöz, Bilkent Üniversitesi

Ersin Er, Hacettepe Üniversitesi

Cengiz Erbaş, Aselsan

Nadia Erdoğan, İstanbul Teknik Üniversitesi

Çiğdem Gencel, Blekinge Institute of Technology

Yenal Göğebakan, Cybersoft

Haluk Gümüşkaya, Haliç Üniversitesi

Kayhan İmre, Hacettepe Üniversitesi

Oya Kalıpsız, Yıldız Teknik Üniversitesi

Ümit Karakaş, İstanbul Kültür Üniversitesi

Geylani Kardaş, Ege Üniversitesi

Halit Oğuztüzün, Orta Doğu Teknik Üniversitesi

İpek Özkaya, SEI-CMU

Özcan Öztürk, Bilkent Üniversitesi

Ali Özzeybek, Meteksan Savunma

Hasan Sözer, Twente University

Ediz Şaykol, Havelsan

Ayça Tarhan, Hacettepe Üniversitesi

Murat Tanık, Alabama Üniversitesi

Bedir Tekinerdoğan, Bilkent Üniversitesi

Cengiz Toğay, Çanakkale Onsekiz Mart Üniversitesi

Yasemin Topaloğlu, Ege Üniversitesi

Eray Tüzün, Havelsan

UYMK YÖNLENDİRME KURULU

Naci Akkök, Oslo Üniversitesi

Mehmet Akşit, Twente Üniversitesi

Levent Alkışlar, Aselsan

Semih Çetin, Cybersoft

Oğuz Dikenelli, Ege Üniversitesi

Ali Doğru, Orta Doğu Teknik Üniversitesi

Oya Kalıpsız, Yıldız Teknik Üniversitesi

Bedir Tekinerdoğan, Bilkent Üniversitesi (Başkan)

Yasemin Topaloğlu, Ege Üniversitesi

Davetli Konuşmacılar

Agility and Architecture: Why and how they can co-exist?

Muhammad Ali Babar,

IT University of Copenhagen, Denmark

Abstract

Agile approaches have gained popularity as a mechanism for reducing cost and increasing ability to handle change in dynamic market conditions. However, there is a significant concern about the role and importance of the issues related to the software architecture of a system being developed using agile approaches. There seems to be a lot of misconceptions about the value of sound architecture practices among agile followers. Proponents of architecture-based methods have their own doubts about the scalability of agile approaches. All these misconceptions lead to many people believe that there exists a dichotomy between agile and architecture values and principles and both can't or shouldn't co-exist. However, there is also a growing interest in separating the facts from myths about the necessity, importance, advantages and disadvantages of combining agile and architectural principles and approaches. This talk explores some of the motivators of the questions like, how can organizations make their development approaches agile without compromising the due role and importance of software architecture. It then discusses a few industrial cases aimed at combining agile and architectural approaches. Finally, it presents some practical strategies that can help answer questions like "Why and How can Agile and Architecture co-exist?

Bio

Dr. Muhammad Ali Babar is an Associate Professor at IT University of Copenhagen, Denmark. He has authored/co-authored more than 100 peer-reviewed research papers in software engineering journals, conferences, and workshops. He has also co-edited a book, software architecture knowledge management: theory and practice. His editorial assignments include co-guest editing five special issues (to be) published in IEEE Software, JSS, ESEJ, SoSyM and REJ on software architecture related topics. Apart from being on the program committees of several international conferences, Dr. Ali Babar is also chairing the program committees of several conferences such as ECSA2010 (Chair), PROFES2010 (Co-Chair), and ICGSE2011 (Co-Chair). He is a member of steering committees of WICSA, ECSA, and ICGSE. He has presented tutorials in the areas of software architecture and empirical approaches at various international conferences including ICGSE09, XP08, ICSE07 and WICSA07. Prior to joining R&D field, he worked as a software engineer and an IT consultant for several years in Australia. He obtained a Ph.D. in Computer Science and Engineering from the school of computer science and engineering of University of New South Wales.

Evaluating The Software Architecture Competence of Organizations

Paul Clements
Software Engineering Institute,
USA

Abstract

An organization is architecturally competent if it has the ability acquire, use and sustain the skills and knowledge necessary to carry out architecture-related practices that lead to systems that serve the organization's business goals. Previous work in architecture has concentrated on the technical: methods and tools for creating, analyzing, and using architecture. However, a different perspective recognizes that these activities are carried out by people working in organizations, and those people and organizations can use guidance in doing their jobs better with respect to consistently producing high-quality architectures. This paper presents some principles of architecture competence, and describes four models that aid in explaining, measuring, and improving the architecture competence of an individual or an organization with respect to these principles, The principles are based on a set of fundamental beliefs about software architecture. The models are based on (1) the duties, skills, and knowledge required of a software architect or architecture organization; (2) human performance technology, an engineering approach applied to improving the competence of individuals; (3) organizational co-ordination, which explains how people and units in an organization share information; and (4) organizational learning, which explains how organizations acquire, internalize, and use knowledge to improve their performance.

Bio

Dr. Paul Clements is a senior member of the technical staff at Carnegie Mellon University's Software Engineering Institute, where he has worked since 1994 leading or co-leading projects in software product line engineering and software architecture documentation and analysis. Clements is the co-author of three practitioner-oriented books about software architecture: "Software Architecture in Practice" (1998, second edition 2003), "Evaluating Software Architectures: Methods and Case Studies" (2001), and "Documenting Software Architectures: View and Beyond" (2002, second edition 2010). He also co-wrote "Software Product Lines: Practices and Patterns" (2001), and was co-author and editor of "Constructing Superior Software" (1999). In addition, Clements has also authored dozens of papers in software engineering reflecting his long-standing interest in the design and specification of challenging software systems. In 2005 and 2006 he spent a year as a visiting faculty member at the Indian Institute of Technology in Mumbai. He was a founding member of the IFIP WG2.10 Working Group on Software Architecture.

Yazılım Mimarisi Tasarımı - I

Görünümler ve Ötesi Yaklaşımıyla Radar Yazılım Mimarisi Dokümantasyonu Tecrübeleri

Ali Özzeybek[1], Mahmut Devrim Tokcan[2], Murat Tuncer[3]
[1,2,3] Meteksan Savunma San. A.Ş., 06800, Bilkent, Ankara
[1] aozzeybek @meteksansavunma.com.tr, [2]dtokcan@meteksansavunma.com.tr,
[3] mtuncer@meteksansavunma.com.tr

Özet. Bu bildiride Meteksan Savunma ve Bilkent Üniversitesi ortaklığı tarafından geliştirilen Milimetrik Dalga Radar (MILDAR) Sisteminin yazılım mimarisinin Görünümler ve Ötesi (Views and Beyond) yaklaşımıyla yeniden dokümante edilmesi sırasında edinilen tecrübeler paylaşılmıştır. Bildiride öncelikle Görünümler ve Ötesi yaklaşımında yer alan kavramlar anlatılmıştır. Bu kısımda özellikle Yazılım Mimarisi alanında eksikliği hissedilen Yazılım Mimarisi Sözlüğü oluşturulmaya başlanmıştır. Geliştirilen MILDAR sisteminin paydaşları ve ilgileri tespit edilerek, bu paydaşların ihtiyaç duyduğu/duyabileceği görünümler Görünümler ve Ötesi yaklaşımı ile ortaya konulmuştur. Farklı bakış açıları ile ele alınan bu çalışmada, kodlama birimi perspektifi, çalışma zamanı perspektifi, yerleşim perspektifi kullanılarak bütünsel ve tümleşik bir yazılım mimarisi elde edilmiştir.

1 Giriş

"Kara Hedef Angajmanı için Milimetrik Dalga Radar Teknikleri Geliştirilmesi" Projesi, kısa adıyla MILDAR Projesi, Meteksan Savunma Sanayii A.Ş ve Bilkent Üniversitesi işbirliğiyle gerçekleştirilen, 1007 programı çerçevesinde TÜBİTAK tarafından desteklenen ve Savunma Sanayii Müsteşarlığı tarafından yürütülen bir ARGE projesidir. Projede, helikopter platformları göz önüne alınarak, prototip bir radar sistemi tasarlanmaktadır. Hedef tespit, takip ve sınıflandırma işlevlerini, Darbe-Doppler, Sentetik Açıklıklı Radar (SAR), Ters Sentetik Açıklıklı Radar (ISAR) ve Arazi Profili Oluşturma uygulamalarını bir bütün olarak barındıran bu radar sistemi, ülkemizde milimetre dalga bandında çalışan ilk prototip radar sistemi olacaktır [1][2].

MILDAR Sistemi yazılım geliştirme çalışmaları, ihtiyaç duyulan kısıtlı sayıdaki mimari görünümle başlamış ancak sistem tasarımı ve yazılım tasarımındaki değişikliklerle daha fazla görünüm ihtiyacı ortaya çıkmıştır. Literatürde mevcut olan 4+1[3], Siemens 4 Görünüm Modeli[4] ve Görünümler ve Ötesi[5] gibi farklı mimari bakış açısı yaklaşımları incelenmiştir. Bunlardan Görünümler ve Ötesi Yaklaşımı güncel yaklaşımlardan biri olması ve genişlemeye uygunluğu nedeniyle seçilmiştir. Görünümler ve Ötesi yaklaşımı ile MILDAR Sisteminin Yazılım Mimari Dokümantasyonu yeniden yapılandırılmıştır. Bu çalışma kapsamında paydaşlar ve ilgilerinin somut olarak ortaya konulması ve alan analizinin bir alan modeli ile ortaya konulması da hedeflenmiştir.

Bildiride öncelikle yazılım mimarisi paydaşları ve ilgileri ortaya konulacaktır. Daha sonra alan analizi sonucu ortaya çıkan alan modeli tanımlanacaktır. Alan modeli Radar

alanını ve MILDAR Sistemini genel olarak anlamak açısından önemlidir. Alan modelinden sonra MILDAR Sisteminde paydaşların ilgilerine yönelik olarak hazırlanan görünümler sunulacaktır. Bildirinin sonuç kısmında ise elde edilen tecrübeler paylaşılacaktır.

Bildiride yer alan şekiller, mimari dokümantasyon çalışması kapsamında hazırlanan dokümandan alınmıştır, ancak şekillerin tamamı bu bildiride verilmemiştir.

2 Görünümler ve Ötesi Yaklaşımı

Yazılım mimarisi, yazılım mimarı ya da mimari ekibi tarafından paydaşların ilgilerine yönelik görünümleri ortaya koymak amacıyla hazırlanır. Bir sistemin yazılım mimarisi, yazılım içeriğinin yapısını veya yapı gruplarını, yazılım bileşenlerinin dışarıya açık özniteliklerini ve bu bileşenlerin kendi aralarındaki ilişkilerini içerir [6]. Yazılım mimarisinin dokümantasyonu, bu mimarinin tanımlanması kadar önemlidir. Mimari dokümantasyonu öncelikle sistemin en üst seviyeden tanımlanmasını sağlar, aynı zamanda paydaşlarla iletişim için çok iyi bir araçtır. Bir yapının ana planı olarak değerlendirilip detaylı analiz için gerekli temel bilgileri içerir. Yazılım geliştirme sürecinde rehber olarak kullanılabilir, iş ataması gibi organizasyon ihtiyaçlarının karşılanmasında faydalı olabilir.

Mimari tasarım modelleme ve dokümantasyonu ile ilgili anahtar kavram hiç şüphesiz görünüm (View) kavramıdır. Günümüzün büyük ölçekli yazılım sistemleri basit ve tek boyutlu olarak ifade edilemezler. Yazılım mimarisinin görünümlerini, farklı paydaşların ilgileri şekillendirir. Paydaş (Stakeholder) kavramı aşağıdaki şekilde tanımlanmıştır:

[Bir sistemin paydaşı] ilgili sistemle ilgisi ya da beklentileri olan birey, ekip ya da kurumdur [7].

Yazılım mimari modelleri, paydaşların farklı ilgilerini karşılamak amacıyla birden fazla görünüm içerir. Her bir görünüm kendine özgü bir notasyon kullanılarak tanımlanır. Görünümler paydaşların ihtiyaçları doğrultusunda modelleme notasyonlarını içeren "görünüm bakış açısı" (viewpoint) kullanılarak tanımlanırlar.

Bugüne kadar yazılım mimari dokümantasyonu için farklı mimari bakış açıları önerilmiştir. İlk yaklaşımlardan birisi de Philippe Kruchten tarafından ortaya atılan ve büyük ölçüde UML kullanılarak gerçeklenen 4+1 yaklaşımıdır [3][8]. 4+1 görünüm yaklaşımında yazılım mimarisi dört ana görünümde anlatılmış ve beşinci bir görünümle bu görünümler birbiri ile ilişkilendirilmiştir.

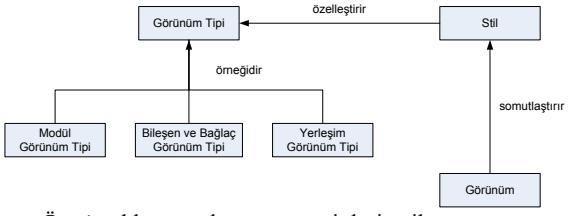
Yazılım mimari dokümantasyonunda giderek artan bir fikir birliği ile çoklu görünüm, ihtiyaç olarak kabul edilmiştir. Çoklu görünüm yaklaşımında, görünümlerin tanımlanmasında açık uçlu, görünümleri belirli bir sayı ve model ile sınırlamayan bir yaklaşım benimsenmiştir. Bu yaklaşımda görünümler paydaşlara ve onların beklentilerine göre belirlenir. Bu fikirle alakalı olarak "bakış açısı" kavramına benzeyen "görünüm tipi" kavramı ortaya atılmıştır [5].

Görünüm tipi, görünümlerin sistematik gruplandırılması olarak değerlendirilebilir. Bu şekilde üç faklı görünüm tipi ortaya çıkarılmıştır.

Modül Görünüm Tipi, sistemin ana yazılım birimlerinin tanımlama ve dokümantasyonunda; Bileşen ve Bağlantılar Görünüm Tipi sistemin çalışan birimlerinin tanımlama ve dokümantasyonunda; Yerleşim Görünüm Tipi, yazılım geliştirme ve çalıştırılma ortamlarının tanımlama ve dokümantasyonunda kullanılır.

Mimari stil veya kısaca stil görünüm tiplerinin özelleşmiş ve belli kalıplara uydurmak için kısıtlanmış şeklidir. Örnek olarak bu çalışmada da kullanılan "Pipes and Filters" Stili Bileşenler ve Bağlantılar Görünüm Tipi'nin özelleşmiş şeklidir. Bileşenler ve Bağlantılar Görünüm Tipi bileşen ve ilişkilerinden bahsederken, "Pipes and Filters" Stili bileşenlerin ve ilişkilerinin nasıl birbiriyle ilişki kurabileceğini kurallarla tarif eder.

Görünümler ve Ötesi yaklaşımı birtakım mimari stilleri tanımlamış olsa da, ihtiyaçlara göre yeni mimari stiller tanımlamak mümkündür. Görünümler stillerden türetilmişlerdir. Görünüm tipleri, stil ve görünüm arasındaki ilişkiler Şekil 1'de gösterilmiştir.



Şekil 1 Görünümler ve Ötesi yaklaşımında görünüm tipleri, stil ve görünüm arasındaki ilişki

Bu çalışma sırasında, ülkemizde üzerinde birlik sağlanan yazılım mimarileri sözlüğünün ihtiyacı oldukça fazla hissedilmiştir. Özellikle Görünümler ve Ötesi yaklaşımında geçen terimler için ortak bir sözlük olmadığı için bu çalışma kapsamında bir mimari sözlük de oluşturulmuştur. Bu çalışma sırasında ortaya çıkan bazı terimler ve Türkçe karşılıkları Tablo 1'de verilmiştir. Bu sözlüğün benzer çalışmalarda genişletilerek kullanılması planlanmaktadır.

Tablo 1 Dokümanda adı geçen terimler ve Türkçe karşılıklarından örnekler

İngilizce	Türkçe
Views and Beyond Approach	Görünümler ve Ötesi Yaklaşımı
Allocation Viewtype	Yerleşim Görünüm Tipi
Module Viewtype	Modül Görünüm Tipi
Component and Connector Viewtype	Bileşenler ve Bağlantılar Görünüm Tipi
Architectural Style	Mimari Stil
Decomposition Style	Ayrıştırma Stili
Uses Style	Kullanım Stili

3 Paydaş Analizi

Paydaşların ve ilgilerinin net olarak ortaya konması görünümlerin ve kullanılacak stillerin belirlenmesi açısından önemlidir. Görünümler kısmında ilgili görünümün hangi paydaşlar için hazırlandığı bilgisi de verilecektir. MILDAR Sisteminin paydaşlarından bazıları ve ilgileri aşağıdaki gibi belirlenmiştir;

Müşteri: Müşteri, projeyi finanse eden ve alıcı makam olarak projeyi yöneten kurum veya kurumlardır. Müşteri sistemin tüm fonksiyonlarıyla belirlenen zamanda ve belirlenen bütçe içerisinde tamamlanmasını bekler. Projenin geliştirilmesi sırasında süreçlere uyulması, dokümantasyon setinin tam olmasına önem verir.

Proje Yöneticisi: Proje Yöneticisi, sistemin belirlenen zaman ve bütçe içerisinde yapılması için proje planının takip edilmesinden sorumludur. Sistemin iş dağılımının yapılmasına önem verir. Proje kapsamında üretilen dokümanların zamanında ve tam olarak çıkması önemlidir.

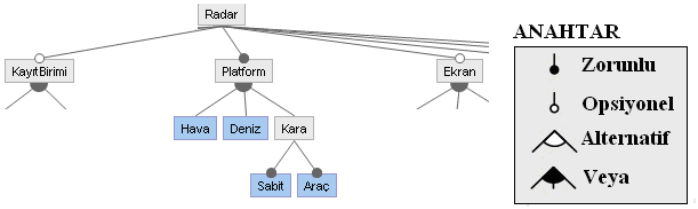
Yazılım Geliştiricileri: Sistemde tüm birimler üzerindeki yazılımların geliştirilmesinden ve algoritmaların kodlanmasından sorumludurlar. Donanım-Yazılım arayüzlerinin net olmasına, sistem senaryolarının tanımlı olmasına önem verirler. Donanımların doğru çalışması yazılım geliştirmenin sorunsuz olması açısından önemlidir.

4 Alan Analizi

4.1 Radar Alan Bilgisi ve Özellik Modeli

Paydaşlar belirlendikten sonra, Radar alanı ve MILDAR Sistemi bir alan modeli ile ortaya konulmuştur. Radar alan analizi sonucunda bir Radar sisteminin zorunlu ve opsiyonel özellikleri incelenerek Şekil 2'deki özellik modeli oluşturulmuştur.

Özellik modeli, Görünümler ve Ötesi yaklaşımında olmamasına karşın Yazılım Ürün Hattı'nda yer alan değişken ve ortak noktaların analizinde kullanılan bir tekniktir [9]. Özellik modeli bir sistemin zorunlu, opsiyonel veya birbiriyle bağlantılı özelliklerini ortaya koymak için hazırlanır. Radar özellik modeli bu doküman kapsamında öncelikle Radar alanını daha iyi anlamak için ortaya konulmuştur. Ayrıca Radar özellik modeli, değişkenlik olan noktaların görülmesini sağlamış ve ileride nihai ürünün varyasyonlarının üretilebileceğini dikkate alarak yazılım mimari çalışmalarının şekillenmesi gerektiğini göstermiştir.

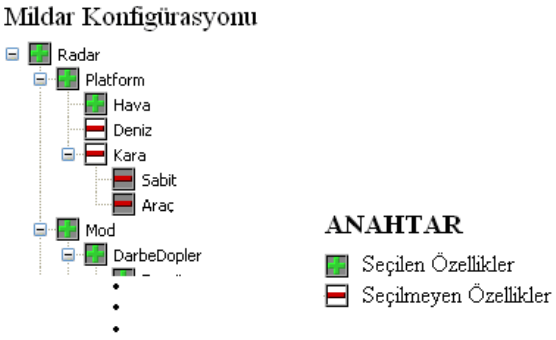


Şekil 2 Radar Özellik Modeli

4.2 MILDAR Konfigürasyonu

MILDAR Sistemi hava platformunda çalışacaktır. Temel olarak Darbe-Doppler, SAR, ISAR ve Arazi Profili Oluşturma uygulama sonuçları bir radar ekranında gösterilecektir. Ayrıca, elde edilen bilgiler analiz amaçlı kaydedilebilecektir.

Oluşturulan Radar özellik modelinde 247 farklı Radar konfigürasyonu ortaya konulabilirken sistem isterleri dikkate alınarak Şekil 3'teki MILDAR konfigürasyonu oluşturulmuştur.



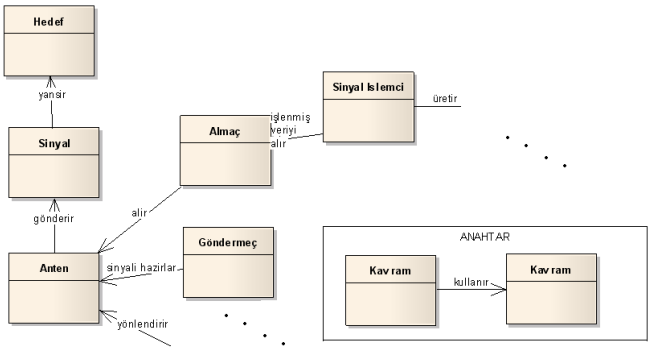
Şekil 3 MILDAR Konfigürasyonu

4.3 Alan Sözlüğü

Alan analizi çalışmasının ürünlerinden biri de alan sözlüğüdür. Alan sözlüğü dokümanın paydaşları tarafından okunduğunda anlaşılabilirliği artırmak amacıyla oluşturulmuştur. Alanda yer alan terimlerin bir sözlük kapsamına alınması doküman boyunca tutarlılığı da sağlamıştır.

4.4 Kavram Görünümü

Kavram görünümü, alan analizi sonucunda ortaya çıkan kavramların birbirleriyle nasıl ilişki kurduğunun anlaşılmasını sağlayan görünümdür. Bu amaçla Radar ve MILDAR Sistemi kavram görünümleri ayrı ayrı hazırlanmıştır. MILDAR Sistemi için oluşturulan kavram görünümünde yer alan bazı kavramlar ve ilişkileri Şekil 4'de verilmiştir.

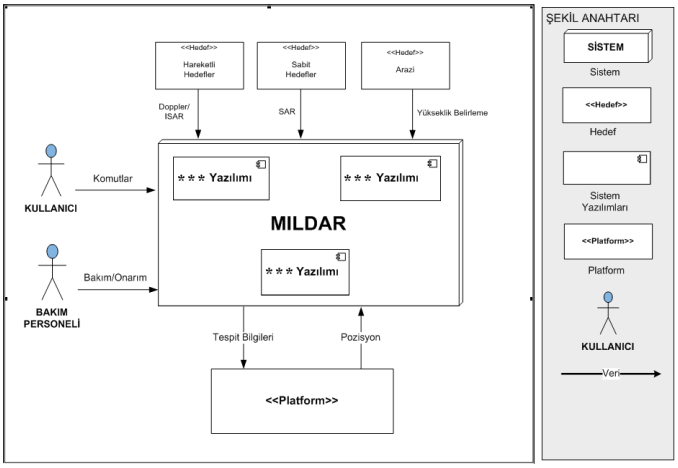


Şekil 4 MILDAR Kavram Görünümü

5 İçerik Görünümü

İçerik görünümü sistemin dış dünyayla ilişkisini göstermek amacıyla kullanılır. Bu görünüm üst seviye olarak sistemi anlattığı için tüm paydaşların ilgilerine yönelik olarak kullanılabilir. Şekil 5'te verilen MILDAR içerik görünümü, sistemin arayüzlerinin ortaya konmasını sağlamıştır. Yapılan çalışma sonucunda MILDAR Sistemi içerik görünümünün üst düzey kaldığı ve yazılım birimleri için bir alt seviye içerik görünümü çizmenin gereklilik olduğu sonucuna varılmıştır ancak bu ayrıntılı görünümler bildiri kapsamında verilmemiştir.

Şekil 5'te, kullanıcı, bakım personeli, platform ve sistemin tespit edeceği hedeflerin sistemin dış arayüzleri olduğu görülmektedir.



Şekil 5 MILDAR İçerik Diyagramı

6 Ayrıştırma ve Kullanım Görünümleri

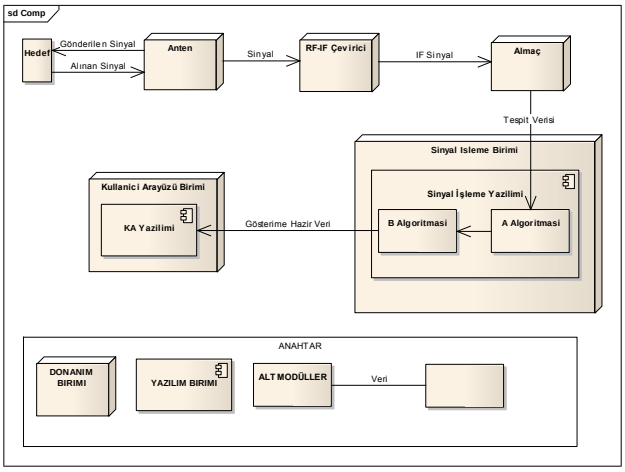
Ayrıştırma ve Kullanım görünümleri kodlama birimi perspektifi kullanılarak oluşturulan görünümlerdir. Ayrıştırma görünümü, modüller ve alt modülleri belirlemek için kullanılan ayrıştırma stilinden türetilmiştir. Kullanım görünümü, ayrıştırma görünümünde belirlenen modüller ve alt modüller arasındaki kullanım ilişkisini ortaya çıkarmak için kullanılmaktadır. Bu görünümlerin bir amacı da ortak modülleri bulmak ve tekrar kullanılabilirliği sağlamaktır. Bu görünümler, yazılım geliştiricileri, yazılım mimarları, yazılım testçileri ve algoritma geliştiriciler tarafından kullanılabilir.

7 Bileşenler ve Bağlantıları Görünümü

Bileşenler ve bağlantıları farklı stillerde gösterilebilir. "Pipes and Filters" Stili MILDAR Sisteminde alınan veriyi ve bu veriyi işleyen birimleri göstermesi açısından en uygun stildir. "Pipes and Filters" Stili daha çok sinyal işleme türü yoğun veri akışına sahip uygulamaların mimarisinde kullanılmaktadır. Bu stil sistem parçalarında iletilen verinin süzülerek daha anlamlı bilginin elde edilmesini sağlamakta ayrıca sistem performansının değerlendirilmesi, zamanlama ve kaynak kullanımı konusunda düzenlemelere imkan verecek etkin bir model ortaya koymaktadır.

Çalışma kapsamında MILDAR Sisteminin bütün modları için "Pipes and Filters" stilinden türetilmiş görünümler oluşturulmuştur.

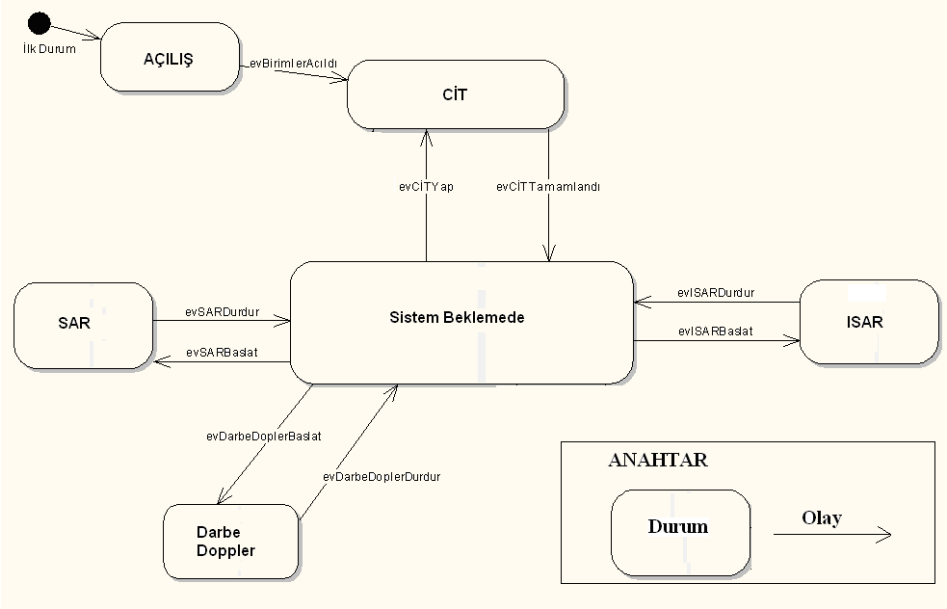
Şekil **6**'da Darbe Doppler modu için bu stilde çizilmiş örnek görünüm yer almaktadır. Bu görünümler verinin nerede işlendiğini ortaya koyduğu için performans iyileştirmelerinde fayda sağlamıştır. Bu görünümler, yazılım geliştiricileri, yazılım mimarları, algoritma geliştiriciler ve sistem mühendisleri tarafından kullanılabilir.



Şekil 6 "Pipes and Filters" Görünümü

8 Davranış Görünümü

Bir sistemin hata dayanıklılığını artırmanın ilk noktası sistemin hangi durumlarda olabileceğinin ve bu durumlardan diğerine nasıl geçileceğinin bilinmesidir. MILDAR Sistemi geliştirilirken ihtiyaç duyulan durum görünümü bu çalışma kapsamında Şekil 7'deki gibi hazırlanmıştır.



Şekil 7 MILDAR Durum Görünümü

9 Kurulum Görünümü

Kurulum görünümü MILDAR'daki yazılımların hangi donanım üzerinde çalışacağını göstermek amacıyla hazırlanmıştır. Kurulum görünümü Alt Yükleniciler, Sistem Tasarımcıları, Yazılım Mimarları, Yazılım Testçileri, Yazılım Geliştiricileri, Donanım Geliştiricileri ve Algoritma Geliştiriciler için önemli bir görünümdür.

Kurulum görünümü, iş yükünün kaynaklar arasında dağıtılabilmesi konusunda iletişim kurulmasını sağlamıştır.

10 Sonuç

Mimari yönelimli yaklaşım uygulamanın temel hedefi paydaşlar arasındaki bilgi paylaşımını artırmak, yazılım geliştirme sürecini desteklemek ve tekrar kullanılabilir yazılım parçalarını tespit edip organizasyondaki verimliliği artırmaktır. Bu hedefe ulaşılabilmesi için mimarinin kapsamlı olarak dokümante edilmesi gerekmektedir. Bu amaçla, mimari dokümantasyon için güncel yaklaşımlardan biri olması ve genişlemeye uygunluğu nedeniyle Görünümler ve Ötesi yaklaşımı benimsenmiştir.

Görünümler ve Ötesi yaklaşımıyla gerçekleştirilen mimari dokümantasyon çalışması öncesinde de MILDAR Sistemi paydaşlarına yönelik görünümler hazırlanmıştır. Ancak, yürütülen önceki çalışmalarda standart ve bütünlük sağlayan görünümlere ve dokümantasyona ulaşılamamıştır.

MILDAR Sisteminin yazılım mimarisinin Görünümler ve Ötesi yaklaşımıyla yeniden dokümante edilmesi sırasında kazanılan tecrübeler aşağıda sıralanmıştır.

- Bu yaklaşım, sadece var olan mimarinin dokümantasyonunu sağlamakla kalmamış, aynı zamanda mimarinin yeniden şekillenip iyileştirilmesine katkıda bulunmuştur.
- Mimari dokümantasyon, alınmış olan mimari tasarım kararlarının daha görünür ve anlaşılır olmasını sağlamıştır.
- MILDAR Sisteminde, bileşen ve bağlantıları görünüm tipinde kullanılan "Pipes and Filters" stili yazılımın işleyiş darboğazlarını tespit etmemizi ve performans iyileştirme konusunda çalışılabilmesini sağlamıştır.
- Ortak bir yazılım mimarisi sözlüğünün ihtiyacı oldukça fazla hissedilmiştir. Bu çalışma kapsamında bir sözlük çalışmasına başlanmıştır.
- Paydaşlarla ortak bakış açıları yakalanarak isterlerin etkin bir şekilde mimariye aktarılması sağlanmıştır.
- Yazılım Mimarisinin kodla entegre olmamasının dokümanı güncel tutmak için ekstra bir çaba gerektireceği değerlendirilmiştir.
- Alan modeli ortaya konulurken, Görünümler ve Ötesi yaklaşımında bulunmayan özellik modeli tekniği kullanılmıştır. Bu sayede geniş kapsamlı olan Radar alanında MILDAR Sisteminin hangi özelliklere sahip olduğu daha iyi resmedilmiştir.
- Görünümler ve Ötesi yaklaşımındaki stil kavramı ile mimari tasarım kalıbı [10] kavramları çakışmaktadır. (örneğin: "Pipes and Filters" tasarım kalıbı – "Pipes and Filters" Stili) Bu durumun stillerin anlaşılmasını ve kullanımını zorlaştırdığı değerlendirilmiştir.

Gelecek çalışma olarak; mimari değerlendirme yöntemleri ile MILDAR Yazılım Mimarisinin fonksiyonel ve kalite açısından değerlendirilmesinin yapılması planlanmaktadır [11]. Ayrıca, Model Güdümlü Yaklaşım kullanılarak oluşturulacak görünümlerin mimari değişikliklere daha kolay adapte edilebilmesinin sağlanabileceği değerlendirilmektedir [12].

Kaynakça

1. Kaderli, G., Şansal, M., Özzeybek, A., ve Tuncer, M., MILDAR Sisteminde Yazılım Tasarım ve Geliştirme Yaklaşımları, ASES 2010.
2. Skolnik, M., Introduction to Radar Systems, Third Edition, McGraw-Hill, New York, 2001
3. Kruchten, P., The 4+1 view model of architecture. IEEE Software, 12(6):42–50, 1995.
4. Hofmeister, C.; Nord, R.; Soni, D. Applied Software Architecture, Reading, MA: Addison-Wesley, 2000
5. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R.,ve Stafford, J., Documenting Software Architectures: Views and Beyond. Addison-Wesley, 2002.
6. Bass, L., Clements, P., ve Kazman, R. Software Architecture in Practice, second edition, Addison-Wesley 1998.
7. Institute of Electrical and Electronics Engineers. Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE Std 1471-2000). New York, NY: Institute of Electrical and Electronics Engineers, 2000.
8. Kruchten, P., The Rational Unified Process: An Introduction, Second Edition. Addison-Wesley, Boston, MA, USA, 2000.
9. Kang, K., Lee, J., ve Donohoe, P. Feature-oriented product line engineering. IEEE Software 19, 4 (July/August 2002), 58–65
10. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Pattern-Oriented Software Architecture Volume 1: A System of Patterns, 1996
11. Clements, P., Kazman, R., ve Klein, M., Evaluating Software Architectures: Methods and Case Studies, Addison-Wesley, 2001.
12. Stahl, T., Völter M., Model-Driven Software Development: Technology, Engineering, Management. Wiley 2006

Veri Erişim ve Yönetim Kütüphanesinin Servis Tabanlı Mimari ile Tasarlanması

Hande Doğan Köseoğlu[1], Serap Bozbey[2]

[1,2] Aselsan A.Ş., Macunköy, Ankara

[1] hdogan@aselsan.com.tr, [2]bozbey@aselsan.com.tr

Özet. Servis Tabanlı Mimari (STM) ile uyumlanabilir, tekrar kullanılabilir servisler yaratmak daha kolay ve tanımlı hale gelmiştir. Gelişen ve değişen gereksinimler ve yazılım geliştirme yaklaşımları ile birlikte daha önce geliştirilmiş olan bileşenler servis tabanlı mimari yaklaşımı ile tekrar ele alınabilmektedir. Veri Erişim ve Yönetim (VERY) Kütüphanesi; verilerin veri tabanında saklanması, veri tabanı işlemlerinin yapılması ve kullanıcıya gösterilmesi işlemlerini uygulamadan bağımsızlaştırmayı hedefleyen bir ara katman yazılımıdır. Servis tabanlı yaklaşımlarda öngörülen minimum bağımlılık ilkesiyle VERY Kütüphanesi için tasarım iyileştirilmelerine gidilmiştir.

Bu bildiride VERY Kütüphanesi mimarisinin STM'ye uyumlu hale getirilmesi ortaya konmakta ve STM'nin değişen gereksinimlere cevap vermede sağladığı kolaylıklar anlatılmaktadır.

1 Giriş

Yazılım mimarilerinde, üretkenliği ve kaliteyi artırmak, ürünün dağıtıma hazır hale gelme süresini azaltmak için her seferinde ürünleri tek başına düşünmek yerine ürün ailesi yaklaşımını benimsemek gerekmektedir [1]. Ürün ailesi yaklaşımı ile yazılım geliştirmede ilgilerin ayrılması esastır. Bileşen tabanlı mimarilerde bu ilgiler bileşenlere dağıtılırken servis tabanlı mimarilerde bu dağıtımın servisler bazında olduğu görülmektedir. Her iki yaklaşım da ürün ailesi geliştirme yaklaşımını mimari açıdan farklılıklar göstererek desteklemektedir.

Bileşen, diğer bileşenlerle beraber belirli bir işlevin gerçekleştirimini sarmalayan yapıya verilen isimdir. Bileşenler hizmetlerini dışarıya standardı belirli olmayan arayüzler üzerinden sunarlar.

Servis, üzerinde anlaşılmış belirli bir arayüz (kontrat) aracılığıyla tanımlanmış işlevleri gerçekleyip yerine getiren ve diğer servislerle sadece bu kontrat üzerinden iletişim kuran bileşen olarak tanımlanmıştır [2]. Servis tabanlı yazılım geliştirmede, kullanıcılar bu kontratlar sayesinde servislere bağımlı olmamakta ve aynı arayüzü gerçekleyen servis sağlayıcılar birbirlerinin yerine geçebilmektedir [2]. Servisler için bu şekilde bir

kontrat sunmak, servislerin bulunmasına, seçilmesine, bağlanmasına ve birleştirilmesine olanak sağlamaktadır [4].

STM hem bir mimari hem de bir geliştirme modeli olarak düşünülebilir [3]. Servisler, bu servislerin tanımları ve servis yaşam döngüsü işlevleri servis tabanlı yazılım geliştirmenin temelini oluşturur [4]. Servis tabanlı yazılım geliştirmeyi oluşturan mimari elemanlar; kontrat, tekrar kullanılabilir bileşenler, kontratlar üzerinden anlaşmayı sağlayan bağlayıcı, çalışma ortamı ve bağlamı olarak tanımlanabilir. Bu elemanlar sayesinde servisler dağıtımı yapılabilir, taşınabilir, güvenilir, hizmet sürekliliği sağlanabilir ve birbirleriyle uyumlu çalışabilir kılınmıştır [2]. Ortamdaki servisler, birbirlerinin bağlamlarından bağımsız olarak çalışmaktadır. Herbir servis bir kara kutu olarak (bileşenin işlevi nasıl yerine getirdiği önemsenmeden sadece hangi çıktıyı üreteceği ile ilgilenilir) düşünülmelidir. Bu servisler aynı çalışma ortamında bulunabileceği gibi, farklı ortamlardan da hizmet verebilirler [3].

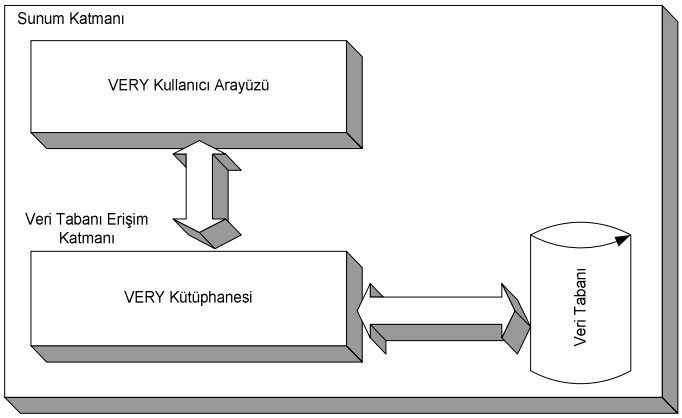
Bileşen tabanlı mimari ile servis tabanlı mimari arasındaki temel farklılık ise; STM'de servisler hizmetlerini belirli kontratlar aracılığı ile bağlamdan bağımsız olarak sunabilirler. Bileşen tabanlı mimaride arayüz tanımlarının standart olmamasından dolayı bileşeni kullanan ve bileşen arasındaki soyutlama daha az düzeyde olabilmektedir.

Bu bildiride, daha önceden bileşen tabanlı mimari esas alınarak geliştirilmiş olan VERY kütüphanesinin, servis tabanlı olarak tekrar ele alınışı aktarılmıştır. Bildiride bileşen tabanlı mimari ile gerçeklenen dönem Faz-1 olarak anılırken, STM'ye geçişin yapıldığı dönem ise Faz-2 olarak anılacaktır. Bildirinin ikinci bölümünde kütüphanenin önceki kullanımı anlatılırken, üçüncü bölümünde STM ile yeniden tasarlanışına ve STM'nin kütüphaneye olan katkılarına yer verilmiştir.

2 Servis Tabanlı Mimariden Önceki Kullanım

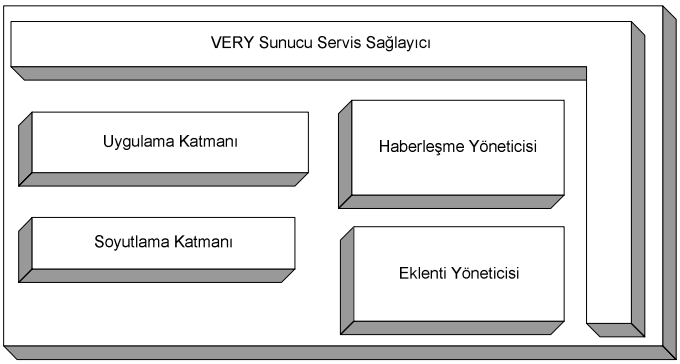
VERY kütüphanesi ilk fazında bileşen tabanlı ve katmanlı mimari kullanılarak geliştirilmiştir [5]. Geliştirilen kütüphanenin esnek ve modüler bir veri erişim ve yönetim mimarisine sahip olması hedeflenmiştir.

Birinci faz VERY mimarisi, veri tabanı erişim ve sunum katmanlarından oluşmuştur (Şekil 1). Şekilde "VERY Kullanıcı Arayüzü" olarak gösterilen bileşen sunum katmanı, "VERY Kütüphanesi" olarak gösterilen bileşen ise veri tabanı erişim katmanıdır.



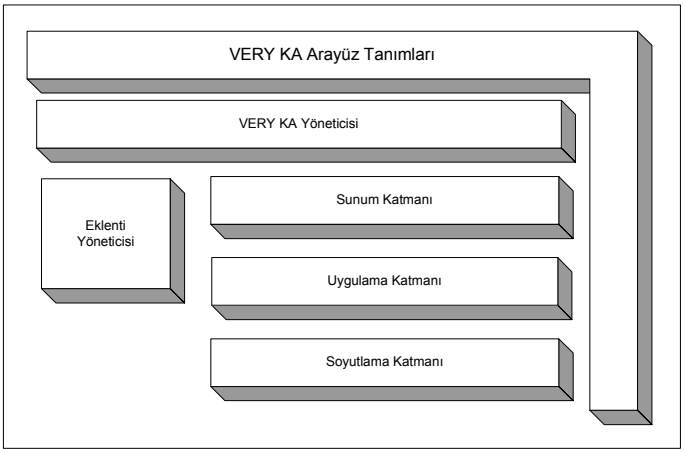
Şekil 1. Faz-1 VERY Mimarisi

Veri tabanı erişim katmanında yer alan VERY Sunucu Kütüphanesi, kendini kullanan uygulamaları veri tabanından soyutlayan ve veri tabanı işlevlerini uygulamalara sunan bir ara katman yazılımıdır. VERY Sunucu Kütüphanesi veri kaynağı olarak sadece veri tabanı ile çalışabilmektedir. Bu kütüphane, veri tabanına bağlantı kurma, ekleme, silme, güncelleme, sorgulama gibi genel veri tabanı işlevlerini, "Hibernate" altyapısını kullanarak gerçekleştirir. (Şekil 2)



Şekil 2. Faz-1 VERY Sunucu Kütüphanesi Yazılım Mimarisi

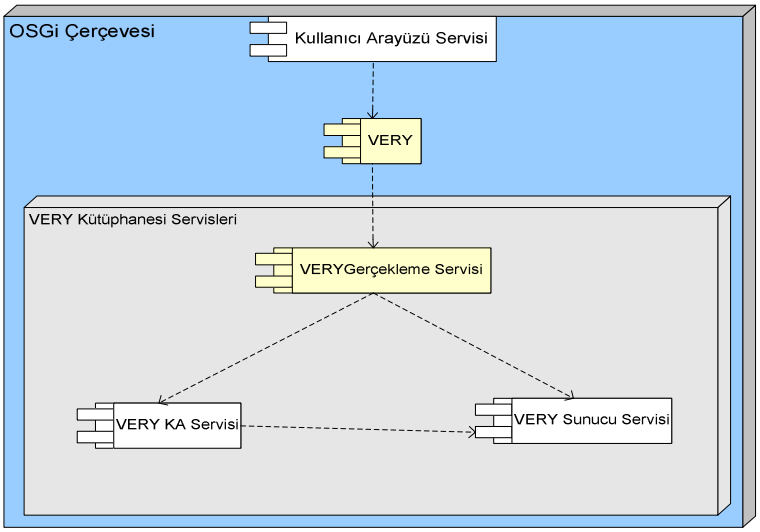
Sunum katmanında yer alan VERY KA (Kullanıcı Arayüzü) Kütüphanesi ise, verilerin sunulduğu grafiksel kullanıcı arayüzlerini ortak bir dilde sunmak ve uygulamaların bu konudaki genel ihtiyaçlarını karşılamak üzere hazırlanan bir yazılımdır. VERY KA Kütüphanesi, VERY Sunucu Kütüphanesini kullanarak verilere erişir ve sunduğu kullanıcı arayüzleri sayesinde veriler üzerinde işlem yapılmasını sağlar. VERY KA Kütüphanesi, eklenti mimarisini destekler ve uygulamalara özel yeteneklerin eklenmesine olanak sağlar. Verilerin kullanıcı arayüzünde gösterilmesi için gerekli bilgileri (gösterilecek veri alanları ve özellikleri) bu amaç için hazırlanmış yapılandırma dosyalarından alır. (Şekil 3)



Şekil 3. Faz-1 VERY KA Kütüphanesi Yazılım Mimarisi

3 Servis Tabanlı Mimariye Geçiş ve Kütüphaneye Kazandırdıkları

STM'nin temelini, bağımlılığı en aza indirgenmiş servisler oluşturur. VERY Kütüphanesi Faz-2 mimarisinin temeli de buna dayanmaktadır. VERY Kütüphanesi, servis tabanlı olarak ele alınırken OSGi (Open Services Gateway Initiative) çerçevesinin bir gerçekleştirimi olan "Equinox" çalışma ortamı kullanılmış ve kütüphane Faz-1'de de olduğu gibi Java dilinde gerçekleştirilmiştir [7,8]. Kütüphane kapsamında gerçekleştirilen tüm servisler aynı çalışma ortamından hizmet vermektedir. Şekil 4'te, kütüphane kapsamında bulunan servisler ve birbirleri ile ilişkisi verilmektedir.



Şekil 4. VERY Kütüphanesi Servisleri

Uygulamalar, VERY Kütüphanesi üzerinden yaptıracakları işler için VERY servisini kullanmaktadırlar. VERY Servisi, uygulamalar tarafından kullanılacak olan ortak arayüz ve bu arayüzlerin ihtiyacı olan diğer sınıfları kapsar.

VERYGerçekleme servisi, VERY servisinin bir gerçeklemesidir. [6]'da belirtildiği gibi bazı servisler diğer temel servisleri kullanarak iş akışını yönetirler, VERY Kütüphanesi kapsamında da VERYGerçekleme servisi kendisine gelen istekleri, VERY KA ve VERY Sunucu servislerine yönelterek iş akışını yönetir.

VERY Sunucu servisi, "Hibernate" alt yapısı üzerinden veri tabanı ile ilgili temel işlemleri (ekleme, silme, güncelleme vs.) yerine getirir. VERY KA servisi ise tabloların liste olarak gösterilmesinden ve liste üzerindeki işlemlerin yerine getirilmesinden sorumludur.

VERY Kütüphanesi kapsamında, STM'ye geçiş ile elde edilen kazanımlar aşağıdaki alt başlıklarda ele alınmıştır.

3.1 Kütüphane Dış Arayüzlerinin Ortaklanması ve İzole Edilmesi:

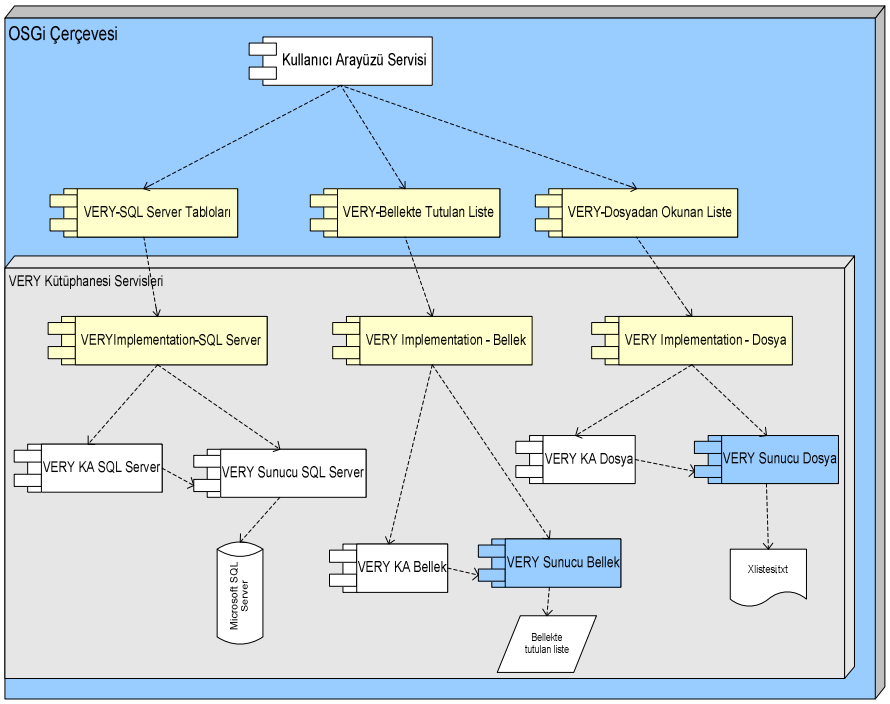
STM'de, servislere erişim kontratlar üzerinden yapılmaktadır. VERY Kütüphanesi için de Faz-2 mimarisinde uygulamaların kullanabileceği bir arayüz hazırlanmıştır. Bu arayüz VERY servisi içerisinde yer almaktadır. OSGi çerçevesi arayüz yönetimi ile de diğer servislerin yardımcı sınıflara doğrudan ulaşımı engellenmiştir. Bu sayede kütüphanenin Faz-1 çalışmalarında bir problem olarak karşılaşılan erişilmesi istenmeyen paketlere erişim de engellenmiştir.

Faz-1'de VERY Kütüphanesi tarafından yapılacak olan işlemler için istekler, hem VERY KA hem de VERY Sunucu kütüphanesine gönderilerek işlevlerin akışı uygulamalar tarafından yönetilmekte idi. Faz-2 mimarisi ile birlikte VERY Kütüphanesinin dış uygulamalara olan arayüzleri de ortaklanmış ve tek bir arayüzde toplanmıştır. Böylece uygulamalar isteklerini ortak arayüze yöneltmekte ve işlevlerin akış kontrolü bu arayüz üzerinden gerçeklenmektedir. Bu sayede VERY Kütüphanesi iç işleyişinden, uygulamalar daha bağımsız hale getirilmiştir.

3.2 Çoklu Veri Kaynağı

Daha önce de belirtildiği gibi kütüphanenin Faz-1 çalışmalarında, uygulamalara sadece tek bir veri kaynağından hizmet sunulabiliyordu. Servis tabanlı yazılım geliştirme ortamının desteği ile servislerin haberleşme noktası arayüzler olduğundan, veri kaynakları konusunda Faz-2 ile beraber esneklikler getirilebilmiştir.

Faz-1'de sadece veri tabanı kullanımı desteklenirken, yeni mimari ile birden fazla veri kaynağını kullanıma almak mümkün olmuştur. Veri kaynağı olarak sadece veri tabanı değil, VERYSunucu arayüzlerini gerçekleyen herhangi bir servis de kullanıma alınmıştır. Böylece Faz-1'de veri tabanında tutulan tablolar liste olarak gösterilebilirken, Faz-2 ile birlikte bellekte, dosyada vb. ortamlarda tutulan bilgilerde de liste gösterimine geçilebilmiştir. Şekil 5'de birden fazla veri kaynağı olan ortamdaki kullanım senaryosu gösterilmiştir.



Şekil 5. VERY Kütüphanesi Servisleri Kullanım Örneği

Şekil 5'de gösterilen senaryoyu kullanabilmek amacıyla, herbir veri kaynağı için VERY kütüphanesinin yapılandırma dosyasında veri kaynakları isimleri ile belirtilmelidir. Veri tabanı dışında bir veri kaynağı kullanılacağı zaman; uygulamalar, VERY Sunucu arayüzünü gerçekleyen servisleri OSGi çerçevesine yerleştirmelidir. Uygulama tarafından sağlanması gereken servisler şekilde "VERYSunucu Bellek" ve "VERYSunucu Dosya" adıyla geçmektedir. VERY Kütüphanesi yapılandırma dosyasından okuduğu bu isimlere uygun servisleri bularak kütüphanenin işlevlerini gerçekleştirir.

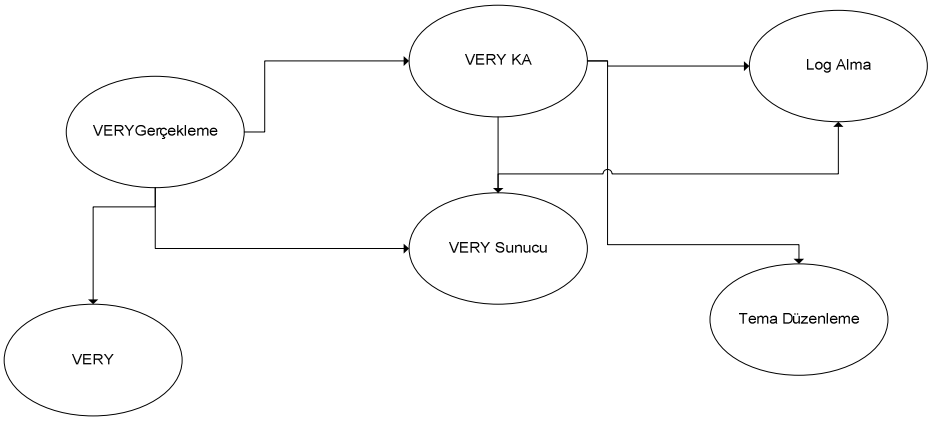
3.3 Eklentiler

VERY Kütüphanesi, listelerin gösteriminde eklenti altyapısını desteklemektedir. Eklenti altyapısının temel amacı; liste üzerinde uygulamaya özel yapılacak işlevleri destekleme ihtiyacını karşılamaktır. Kütüphaneyi kullanan uygulamalar bu tip işlevleri için, liste bazında eklenti servisleri yazarak kütüphanenin işlevselliğini genişletebilirler.

STM'ye geçilmesi ile birlikte daha önceden kütüphane içinde yapılan eklenti yönetimine de gerek kalmamıştır. Eklentiler OSGi çerçevesinde çalışan servislere dönüştürüldüğü için VERY Kütüphanesi'nden eklenti yönetimi kısmı kaldırılmıştır.

3.4 Çalışma Zamanı Yönetimi

STM'nin, VERY Kütüphanesi'ne bir başka katkısı ise; sadece hizmet verebilen servislerle çalışmaya devam edebilmesidir. Bunun için OSGi çerçevesinin servis yaşam döngüsü kontrolleri kullanılmıştır.



Şekil 6. VERY Kütüphanesi ve Diğer Servisler

Şekil 6'da VERY Kütüphanesinin kendi içindeki ve diğer servislerle etkileşimi görülmektedir. VERY Kütüphanesi ilk açılışta ihtiyaç duyduğu bazı servislerin olmaması, çalışmaması durumunda ya da çalışma zamanında herhangi bir anda bu servislerden bazılarının hizmet verememesi, ortama yeni eklenmesi durumlarında kendini yapılandırarak hizmet vermeye devam edecektir. Kendini yapılandırmak ile kastedilen kütüphanenin sunduğu bazı yetenekleri kapatması olacaktır. Örneğin veri tabanına bir kayıt eklerken, VERY-KA servisi hizmet dışı ise, veri tabanı işlemlerinin kullanıcı arayüzü ile ilgili işlevleri yerine getirilemezken (eklenen kaydı liste penceresinde gösterme), hizmet vermeye devam eden VERY Sunucu servisi ile veri tabanına ekleme işlevi yapılabilmektedir.

4 Sonuç

VERY Kütüphanesi Faz-2 çalışmalarında STM'ye geçiş ile birlikte, kütüphaneyi kullanan geliştiriciler açısından hem kütüphanenin kullanımı hem de çalışma zamanı yönetimi kolaylaşmıştır.

Faz-1 çalışmalarında, bir uygulamaya VERY kütüphanesinin özelliklerini eklemek için projenin kendi kodunda daha fazla müdahale yapılması gerekirken, Faz-2 çalışmaları ile birlikte entegrasyon sadece servislerin OSGi çerçevesinden alınarak kullanılmaya başlamasına dönüşmüştür. Bu durum entegrasyon süresini kısaltmıştır. Kütüphaneyi kullanan geliştiriciler, kütüphanenin kendisiyle ilgilenmeyip sadece arayüz tanımları ile kütüphaneyi kullanıma alabilmişlerdir.

VERY Kütüphanesi, Faz-1 çalışmalarında sadece veri tabanı yönetim sistemleri (VTYS) ile beraber çalışabildiğinden, bünyesinde VTYS barındırmayan projelerde kullanıma alınamıyordu. Servislerle iletişimin belirli kontratlarla sağlanması, VERY Kütüphanesi içinde yer alan VERY Sunucu servisinin bu kontratı sağlayan başka bir servisle yer değiştirebilmesini sağlamıştır. Bu durum veri kaynağı olarak veri tabanı kullanan VERY Sunucu servisi yerine değişik veri kaynaklarını kullanabilenen ve VERY Sunucu servisinin kontratını sağlayan diğer servislerin kullanımını mümkün kılmıştır. Faz-2 çalışmaları ile beraber alt taraftaki veri kaynağı değiştirilebilir hale geldiğinden artık daha çok projeye VERY kütüphanesi dağıtımı yapılmaktadır. Faz-1'de performans gereksinimleri nedeniyle bellekte tutulması gereken listeler için bir VERY Sunucu sağlanamamaktaydı ve bu mimaride VERY KA bileşeni VERY Sunucu bileşenine bağlı olduğundan, bellekte tutulan listeler VERY KA kullanılarak gösterilememekte idi. Bu durum, VTYS haricinde herhangi bir veri kaynağı ile çalışan performans gereksinimleri yüksek projelere de dağıtım yapılmasını sağlayarak, yeniden kullanılabilirliği artırmıştır.

Kaynakça

1. Petritsch H., Service – Oriented Architecture (SOA) vs. Component Based Architecture, Şubat 2006
2. Bieber G., Carpenter J., "Introduction to Service-Oriented Programming (Rev. 2.1)", 2001

3. Channabasavaiah K., Holley K., Tuggle E.M., “Migrating to Service-Oriented Architecture”, Nisan 2004

4. Papazoglou M.P., Georgakopoulos D., “Service-Oriented Computing”, Communications of the ACM, Ekim 2003/Vol. 46

5. Atun Y., Bozbey S., “Uyumlanabilir Veri Tabanı Yazılımı Mimari Önerisi”, UYMK 2006

6. Demir Ü., Aktuğ O., “Bir “Mimari Evrim” Hikayesi,”, UYMK 2008

7. OSGi Alliance, www.osgi.org

8. Equinox Project, www.eclipse.org/equinox

Hizmet Odaklı Mimariye Dayanan İş Süreçleri Yönetimi Sistemi

Mine Berker

IBTech A.Ş. TÜBİTAK MAM, TEKSEB, Gebze, KOCAELİ
mine.berker@ibtech.com.tr

Özet. İş dünyasındaki rekabetçi, hızla değişen ortama ayak uydurabilmek, bir işletmenin ayakta kalabilmesi için olmazsa olmazlardan biridir. Hizmet odaklı mimari (SOA – Service Oriented Architecture) nin amacı da, sistemlerin bu hızlı değişime ayak uydurabilecek şekilde tasarlanmasını sağlamak, işlemlerin hizmet olarak sunulmasına izin vermektir[1]. Ancak bu hizmet odaklı yapı sağlandıktan sonra, sunulan bu hizmetlerin arasındaki ilişkinin de dinamik bir şekilde yönetilmesi ihtiyacı doğmaktadır. İş birimlerinin bu hizmetlerin sunulması ile oluşan süreçlerin yönetiminde bilgi teknolojileri (BT) ekipleri ile birlikte hareket edebilecekleri, aynı dili konuşabilecekleri bir yapı, işletmelerin hızlı değişime ayak uydurabilecek, çevik sistemlere sahip olmalarını sağlayacaktır[1]. Bu makalede, hizmet odaklı mimari üzerinde çalışan bir iş süreçleri yönetimi (BPM – Business Process Management) sistemi anlatılmaktadır. Sistem bir iş akış motoru, bir iş kuralları motoru, tanımlanabilir iş yükü dağıtım stratejileri ve süreç ile birlikte takip edilmesi gereken dokümanların yönetilmesi bölümlerinden oluşmaktadır.

1 Giriş

İş süreçleri yönetimi, süreçlere dinamiklik kazandırmak, uçtan uca izlenebilirliği sağlamak ve işletmelerin üretim performansını arttırmak için son yıllarda sıklıkla kullanılan bir yöntemdir. Bunun yanında iş süreçlerinin BT ile uyumunu sağladığı, uygulamalara esneklik kazandırdığı ve değişikliklere hızlı ve kolay cevap verebilmeye imkan tanıdığı için kurumsal çözümlerde tercih edilmektedir[2]. Özellikle hizmet odaklı mimari üzerinde çalışan bir süreç yönetimi sistemi, sunulan hizmetlerin birleştirilmesi, bir arada sunulması ve dış sistemlerle bütünleşmesi konularında mimarinin daha etkin kullanılmasına olanak sağlamaktadır[3].

Piyasada birçok BPM ürünü olmasına rağmen, bu mevcut çözümlerde birçok kısıt göze çarpmaktadır. Öncelikle, mevcut ürünler basit yetkilendirme yapılarını desteklemekte, banka gibi büyük kurumsal işletmelerde bulunan karmaşık örgütsel yapılara uyum sağlayamamaktadır. Benzer şekilde, iş birimlerinin istediği karmaşık süreçleri destekleyecek esnekliğe de sahip değillerdir. Bu ürünlerin, işletme tarafından kullanılmakta olan mevcut uygulama sistemleri ile bütünleşmesi yüksek maliyet gerektirdiği gibi, süreç - doküman bütünleşmesi da yetersiz kalmaktadır.

Bu makalede, Finansbank temel bankacılık sistemi üzerinde çalışan, bankanın örgütsel yapısına ve karmaşık iş gereksinimlerine uyumlu, süreç akışlarıyla dokümanların

birleştirilebildiği bir BPM sistemi anlatılmaktadır. Bu sistem, iş süreçlerinin görsel olarak yönetimini ve izlenmesini, süreç performanslarının takibini ve raporlanmasını mümkün kılmaktadır. Ayrıca sistem hizmet odaklı mimari üzerinde çalışacak şekilde tasarlandığı için, bu mimarinin getirdiği hız ve esneklik avantajından da yararlanmaktadır.

Bu makalenin devamı şu şekilde yapılanmıştır: Öncelikle sistemin ana parçası olan iş akış motoru anlatılmaktadır. Ardından, sürecin ilerlemesine yardımcı olan iş kuralları motoru ve süreç içindeki iş yükünün dağıtılması için kullanılan iş yükü dağıtım stratejilerinden bahsedilmektedir. Sürecin yaşam döngüsü süresince sürece eklenebilen dokümanlar ve mesajlar anlatıldıktan sonra, sistemden alınabilecek raporlardan bahsedilmektedir. Son olarak Finansbank'ta bu sistem üzerinde mevcutta çalışmakta olan süreçlerden örnekler verilerek makale sonlandırılmıştır.

2 İş Akış Motoru

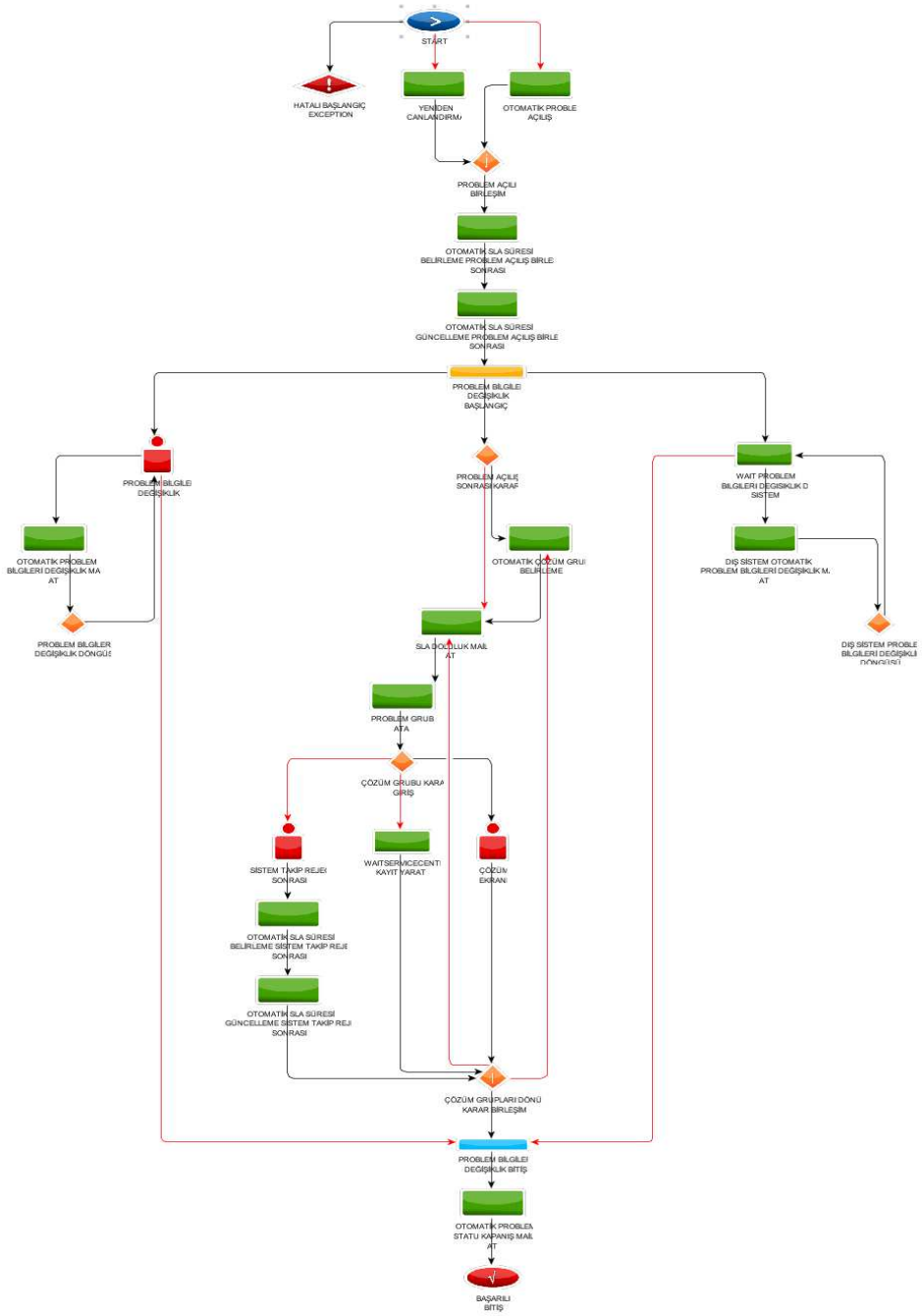
İş süreçleri yönetimi sisteminin temel parçasını, iş akış motoru oluşturmaktadır. Bir süreç çeşitli aktivitelerden ve bu aktiviteler arasındaki geçişlerden oluşacak şekilde tanımlanır. Aktiviteler tiplerine göre ayrılarak, kişiler tarafından gerçekleştirilecek aktiviteler, otomatik gerçekleşecek aktiviteler ya da süreç akışını kontrol etmek amacıyla yerleştirilen aktiviteler olabilir. Şekil 1'de örnek bir süreç akışı görüntüsü bulunmaktadır.

BPM sistemi, Finansbank'ın mevcut temel bankacılık uygulaması üzerinde çalıştığı için, aktivitelerin çalışma mantığı, uygulamanın üzerinde çalıştığı hizmet odaklı mimari altyapısına dayanmaktadır. Sistemdeki her kişi aktivitesi ya da otomatik aktivite, bankacılık uygulaması tarafından sunulan ya da dış sistemlerden sağlanan bir hizmeti çağırmakla görevlidir. Bu şekilde, BPM sisteminin ana uygulama ile birlikte, iç içe çalışması, işlem bütünlüğünün korunmasını sağladığı gibi, kullanıcı önyüzlerinin de mevcut bankacılık kullanıcı arayüzünde bulunmasına, kullanıcıların farklı bir uygulamaya bağlanmadan işlerini tamamlamalarına imkan sağlamaktadır. Şekil 2, kullanıcıların BPM sistemi içindeki işlerini yönetmek için kullandıkları ekranı göstermektedir.

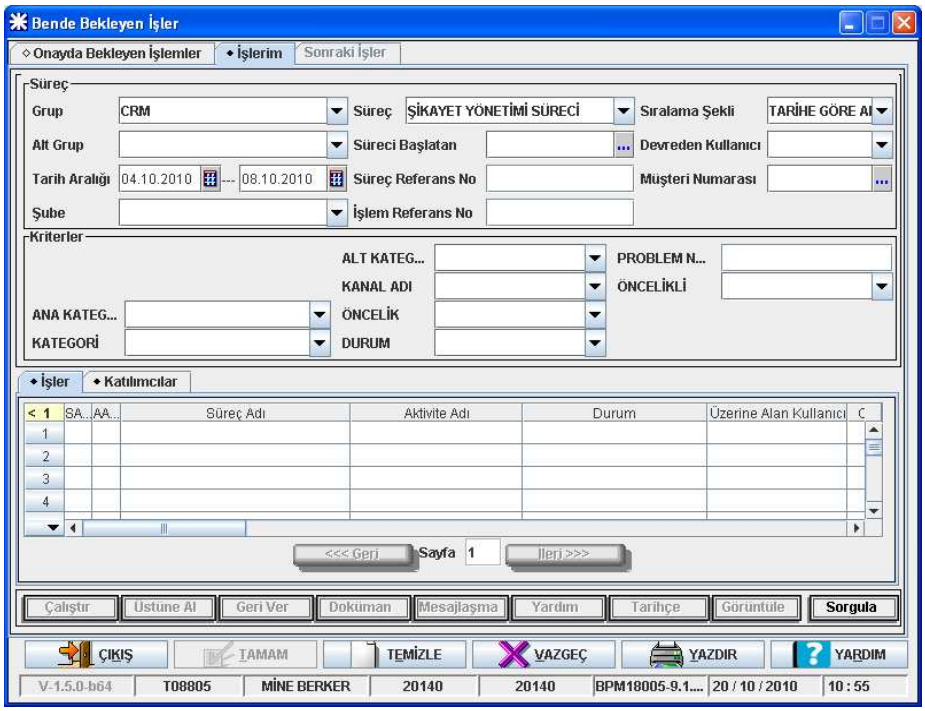
2.1 Yetkilendirme

BPM sisteminde bir süreç başlatılabilmesi için, bu işlemi yapan kişinin ilgili süreci başlatmaya yetkili olması gerekmektedir. Benzer şekilde, başlamış bir süreç içindeki bir kişi aktivitesinin gerçekleştirilebilmesi için de, kullanıcının o süreçteki o aktivite için yetkili olması beklenir. Bu yetki tanımları, bankanın mevcut örgütsel yapısı kullanılarak yapılabildiği gibi, bu yapı dışında, bağımsız kullanıcı grupları oluşturularak da yapılabilir. Böylece, hem karmaşık örgütsel yapılar desteklenmiş, hem de olası yeni yapıları desteklemek için gerekli esneklik sağlanmış olmaktadır.

Yetkilendirmeler tanımlanırken, yetkili kullanıcıların belirlenmesinde, süreç içerisinde daha önce gerçekleşmiş aktiviteleri yapan kullanıcılara referans verilebilmektedir. Örneğin, bir süreçteki onay aktivitesinin yetkileri, aynı süreç içindeki giriş aktivitesini gerçekleştiren kullanıcıyla aynı organizasyonda olan, ya da aynı unvana sahip olan, vs,



Şekil 1. Müşteri Şikayetleri Yönetimi Süreci iş akış görüntüsü



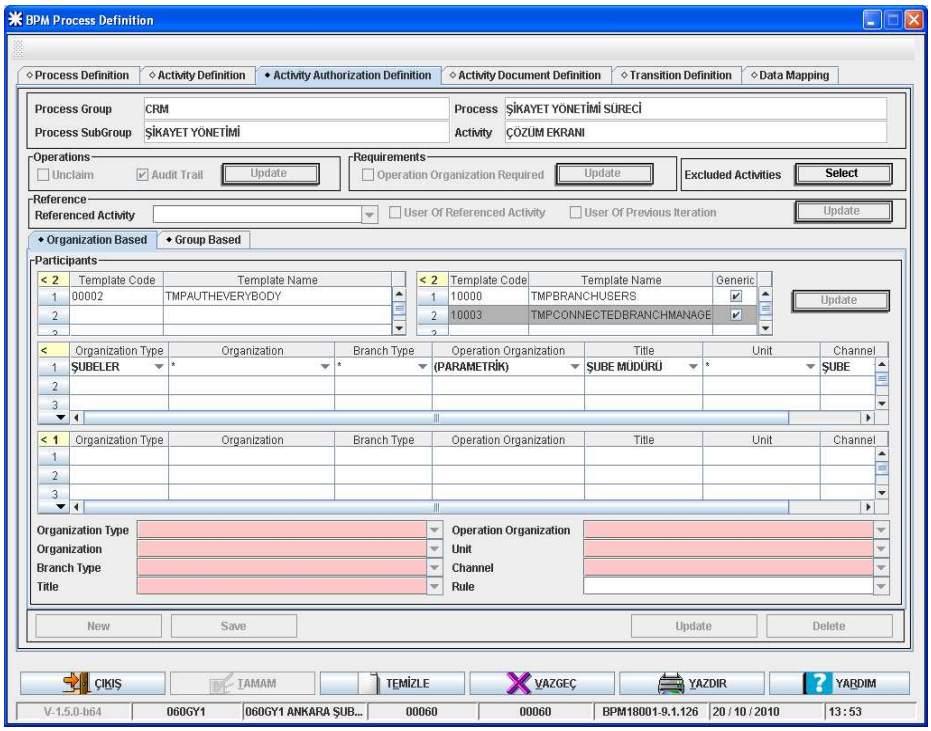
Şekil 2. İş takip ekranı

kullanıcılara verilebilir. Ayrıca onay aktivitesini, giriş aktivitesini yapan kullanıcının yapmaması da sağlanabilir.

Yetki tanımları, birden fazla aktivitede aynı şekilde kullanılacak ise, yeniden kullanılırlık amacıyla bir şablon içinde de tanımlanabilir. Böylece birden fazla yetki tanımı satırı aynı şablon içinde tek bir kez tanımlandıktan sonra, aynı yetkiye sahip olacak aktiviteler direk bu şablon ile ilişkilendirilerek yetkilendirilebilir.

Şekil 3'te, aktivitelere yetkilendirme tanımları yapılan ekran görüntüsü yer almaktadır.

Bir kullanıcı yetkili olduğu bir aktiviteyi gerçekleştirebilmek için, öncelikle bu aktiviteyi üzerine almalıdır. Böylece bu aktivite, yetkili olan diğer kullanıcılar için erişilemez olacaktır. Ancak ihtiyaç olması halinde, özel yetkili kullanıcılar bu aktiviteyi üzerine almış olan kullanıcıdan geri alabilirler. Böylece beklenmedik durumların, sürecin tıkanmasına sebep olmasının önüne geçilmiştir.



Şekil 3. Aktivite yetki tanımlama ekranı

2.2 Paralel Aktiviteler

Bir süreç içindeki bütün aktiviteler sırayla gerçekleşmek durumunda olmayabilir. Eğer iki (ya da daha fazla) aktivitenin birbirlerini beklemelerine gerek yoksa bu aktiviteler paralel ilerleyecek şekilde tanımlanabilirler. Böylece bu iki işi yapacak kullanıcılar, bir diğerinin işini tamamlamasını beklemek zorunda kalmadan aynı süreç üzerinde çalışabilirler. Daha sonra gerçekleşecek bir aktivite, bu iki paralel aktivitenin de tamamlanmış olmasını gerektiriyorsa, süreç her iki paralel kol da tamamlanana kadar ilerlemeyecektir.

Paralel aktivitelerin varlığı, süreç içinde rol alan farklı kullanıcıların gerekmediği sürece birbirlerini beklememelerine imkan sağladığı için, iş süreçlerinin daha kısa sürede tamamlanması mümkün olmuştur.

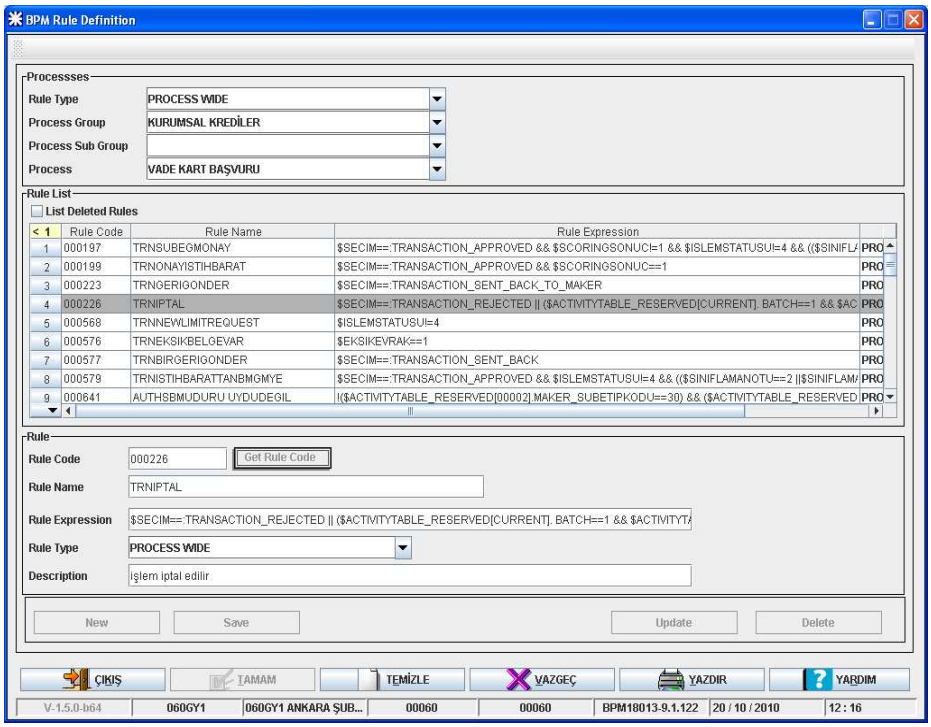
3 İş Kuralları Motoru

Sürecin aktiviteleri arasındaki geçişlerin, her zaman aynı yolu izleyerek çalışmaması, değişik durumlarda değişik aktivitelerin çalışması planlanmış olabilir. Ya da bir

aktivitenin yetkilerinin, çeşitli durumlarda farklı kullanıcı gruplarına verilmesi gerekebilir. Bu esneklik, BPM sisteminde iş kuralları kullanılarak sağlanmıştır.

İş kuralları süreç dışında tanımlanan mantıksal kural ifadeleridir. Sürece ait bilgiler kullanılarak, özel değerlendirme gerektiren bir durumun ayırt edilebilmesini sağlar. Daha sonra bu kurallar süreç içinde kullanılarak süreç akışının duruma göre farklılık göstermesini sağlayabilirler. İş kuralları süreç içinde istenirse aktiviteler arasındaki geçişlere bağlanarak, bir süreçte hangi aktivitenin çalışması gerektiğine karar verebilir, istenirse de yetki tanımı satırlarına bağlanarak, sürecin durumuna göre belirtilen yetki grubunun o aktiviteye yetkili olup olmayacağını belirleyebilir.

Süreç ilerlerken, bu ifadelerin doğru olup olmadığını değerlendiren bir kural doğrulama motoru bulunmaktadır. Bu motor, bir kural ifadesini en küçük mantıksal ifadelerine kadar ayrıştırarak, her parçayı ayrı ayrı değerlendirir ve sonra tekrar bu parçaların sonuçlarını belirtilen şekilde birleştirir. Bu şekilde kuralın doğrulanıp doğrulanmadığını belirler. Kural ifadelerinde, başka kurallar da iç içe kullanılabilir. Örnek bazı iş kuralları Şekil 4'te görülmektedir.



Şekil 4. İş kuralları tanım ekranı

4 İş Yükü Dağıtım Stratejileri

BPM üzerinde yürüyen bir sürece ait bir aktiviteyi gerçekleştirmeye pek çok kişinin yetkisi olsa da, iş yükünü dağıtmak adına farklı gruplardaki kullanıcıların farklı tipte işleri üstlenmesi istenebilir. Bu dağıtımı da sistemin kendisinin yapması tercih edilir. Böyle bir durumda, diğer kullanıcıların aktivite üzerindeki yetkilerini kısıtlamaya gerek olmadan, aktivite ilgili iş grubunun sorumluluk alanına düşecek şekilde tanımlanabilir. Bu aktivite bir kullanıcının sorumluluk alanında olmasa bile, eğer kullanıcının yetkisi varsa, bu aktiviteyi gerçekleştirmesine hiçbir engel olmayacaktır.

Hangi aktivitenin hangi iş grubunun sorumluluğunda olduğuna karar verirken çeşitli stratejiler kullanılabilir. Şu anda sistemde organizasyon tabanlı ve kural tabanlı olmak üzere iki çeşit strateji bulunmaktadır. Ancak gerek olduğunda, başka bir strateji de sisteme kolaylıkla eklenebilir.

Organizasyon tabanlı stratejide, süreci başlatan organizasyona göre, bu süreçteki aktivitelerin hangi iş grubunun sorumluluğunda olacağı belirlenir. Bir iş grubu, birden çok organizasyona bakabilir ve bu organizasyonların her birinden başlatılan süreçlerden sorumlu olabilir.

Kural tabanlı stratejide ise, aktivitenin hangi iş grubunun sorumluluğunda olduğuna, süreç bilgileri kullanılarak çalıştırılacak kurallara göre karar verilir.

5 Doküman Takibi

Bir süreç içerisinde, bu sürece ait sisteme eklenmesi gereken dokümanlar ve müşteri evrakları takip edilebilmektedir. Piyasadaki mevcut BPM ürünlerinin desteklemekte yetersiz kaldığı bu konu, özellikle bankacılık uygulamaları gibi uygulamalarda, kanuni zorunluluklar sebebiyle önem taşımaktadır.

Bir süreç dahilinde sisteme alınması gereken süreç dokümanları ve bu süreç içinde kontrol edilmesi gereken müşteri evrakları, süreç tanım aşamasında sisteme tanımlanır. Hangi dokümanın hangi aktivitede alınabileceği de ayrıca tanımlanmaktadır. Süreç ilerlerken, her aktivitede alınabilecek dokümanlar kullanıcıya listelenir ve kullanıcıdan bu dokümanları sisteme eklemesi istenir. Bazı dokümanlar, süreç için zorunlu ise ve bu şekilde tanımlanmışsa, bunlar sisteme eklenmeden sürecin ilerlemesine izin verilmemektedir.

Süreç devam ederken ve tamamlandıktan sonra, gözlem yetkisi olan kullanıcılar bu süreçte kullanılan, sisteme eklenmiş dokümanları gözlemleyebilmektedirler.

6 Mesajlaşma

Süreç içinde, farklı aktiviteleri gerçekleştiren kullanıcıların birbirleriyle haberleşebilmeleri için bir mesajlaşma yapısı bulunmaktadır. Bu mesajlaşma iki çeşit olabilmektedir:

6.1 İşlemi Başlatana Soru Yönlendirme

Süreç içinde rol alan kullanıcılar, süreç ile ilgili açık olmayan bir konu olduğunu düşündüklerinde, süreç kendi üzerlerinde beklemekteyken, bu süreci başlatan kullanıcıya soru sorabilirler. Süreci başlatan kullanıcı bu sorudan bir otomatik sistem elektronik postası ile haberdar edilir. Soru cevaplandığında, benzer bir elektronik posta ile soruyu soran kullanıcı da bilgilendirilir ve sürecin kaldığı yerden devam etmesi mümkün olur.

6.2 Sürece Not Ekleme

Süreç içinde rol alan her kullanıcı, o süreçle ilgili önemli olduğunu düşündüğü bir konuyu, kendisinden sonra sürece dahil olacak kullanıcıların da bilgisine sunmak için bir not olarak sürece ekleyebilir. Böylece her kullanıcı sürecin mesajlar bölümüne girdiğinde, kendisinden önce sürece eklenmiş notları görebilir ve bu notlardaki bilgiler dahilinde hareket edebilir.

7 Raporlama

Sürecin yaşam döngüsü içinde, başlangıcından bitişine kadar, gerçekleşen her türlü olay, sürece dahil olan kullanıcıların yaptıkları her türlü faaliyet ve sürece eklenen her türlü bilgi, BPM sistemi tarafından kaydedilmektedir. Bu kayıtlar, daha sonra süreçlerin verimliliğinin izlenmesi ve raporlanması için detaylı bir kaynak oluşturmaktadır. Elde edilen raporlara göre, süreçler üzerinde yapılabilecek iyileştirmeler belirlenmekte ve planlanarak kullanıma alınmaktadır.

Bu kayıtlarla, tüm süreç adımları izlenebilmekte, süreç tarihçesi gözlemlenebilmekte ve birim, kullanıcı, süreç ve aktivite bazında performans raporları alınabilmektedir.

8 Canlı Süreçler

Bu makalede anlatılan iş süreçleri yönetim sistemi, Finansbank temel bankacılık uygulamasında bir yılı aşkın süredir kullanılmaktadır. Bu yapı ile bireysel ve kurumsal kredi başvuru süreçleri, dış ticaret akreditif dosya açılış süreci ve çek teminat süreci yeniden modellenmiştir. Alınan sonuçlar, sistemin süreç verimliliğini arttırdığını, kullanıcı iş yükünü azalttığını ve daha kısa sürede daha çok iş üretilmesini sağladığını göstermektedir.

Banka için kritik önemi olan başka bir sürecin, Müşteri Şikayetleri Yönetimi sürecinin de BPM sistemi üzerinde modellenmesi çalışmalarına devam edilmektedir.

9 Sonuç

Bu makalede, Finansbank'ın mevcut temel bankacılık sistemi üzerinde çalışan ve hizmet odaklı mimari altyapısına dayanan bir iş süreçleri yönetimi sistemi anlatılmıştır. Bu sistem, iş süreçlerindeki hızlı değişiklik ihtiyaçlarına ayak uydurabilmek, bu değişimleri bankacılık uygulamasına kolay ve hızlı bir şekilde yansıtabilmek ve süreçlerin verimliliğini izlenebilir kılmak amacıyla tasarlanmıştır. Geçen bir yıllık süre

boyunca sistemden alınan sonuçlar, birçok iş sürecinin iyileştirilmesine katkı sağlamıştır. Finansbank'ın iş süreçlerinin yeniden modellenerek bu sisteme alınması çalışmalarına devam edilmektedir.

Kaynakça

1. Chen Ling, Lu Xin, "Achieving Business Agility by Integrating SOA and BPM Technology" International Forum on Information Technology and Applications, vol. 1, s. 334-337, Mayıs 2009.
2. Dan Luo, Jianhua Jiang, Buyun Sheng, Mingzhong Yang, "Research on Business Process Management System Based on SOA" 9th International Conference on Computer-Aided Industrial Design and Conceptual Design, s. 1108-1111, Kasım 2008.
3. Yang Liu, Enzhao Hu, Xudong Chen, "Architecture of Information System Combining SOA and BPM" International Conference on Information Management, Innovation Management and Industrial Engineering, vol. 1, s. 42-45, Aralık 2008.

Model-Güdümlü Yazılım Mimarisi - II

Akıllı Kart Yazılımlarının Model Güdümlü Geliştirilmesi

Hidayet Burak Sarıtaş[1], Geylani Kardaş[2]

[1,2] Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, 35100, Bornova, İzmir
[1] burak.saritas@kentkart.com.tr, [2]geylani.kardas@ ege.edu.tr

Özet. Akıllı kartlar kendi içerilerinde gömülü bir işlemci ve bellek bulundurabilen, farklı türlerdeki verileri güvenli bir şekilde saklama ve bu bilgileri kullanabilme özelliğine sahip entegre devrelerdir. Günlük hayatta tanımlama, doğrulama ve veri güvenliği gerektiren birçok ticari işlemi en az seviyede kullanıcı etkileşimi gerektirerek sağlayan akıllı kartlar, boyutları dolayısıyla da kullanım alanlarını genişletmektedirler. Bu özelliklerine rağmen, akıllı kart yazılımlarını geliştirmek, alt seviye iletişim yapıları, donanımsal sebepler ve geliştirme aşamasında kullanıcıya getirdiği bazı kısıtlar nedeniyle, geliştiriciler için sıkıntılı bir hal almaktadır. Bu çalışmada, akıllı kart yazılımlarının otomatik, daha basit bir şekilde ve küçük kod hataları engellenerek üretilmesini sağlayan model güdümlü bir yazılım geliştirme yöntemi tanıtılmaktadır. Akıllı kart standartları göz önünde bulundurularak oluşturulan platform bağımsız bir üstmodelden model dönüşümleri sonrası iki önemli akıllı kart platformu için yazılım modelleri ve otomatik kod üretimi sağlanmaktadır.

1 Giriş

Akıllı kartlar günümüzde, telekomünikasyondan ulaşıma, kredi kartı endüstrisinden sağlık kuruluşlarına kadar geniş bir yelpazede kullanılmaktadır. Akıllı kartların tercih edilmesindeki başlıca neden taşınabilirliktir. ISO/IEC 7816 [1] ile fiziksel karakteristikleri ve iletişim altyapıları standartlaştırılan akıllı kartların mikroişlemcisi olması, veriyi işleme ve saklama özelliği bulunması sebebiyle taşınabilirlik ayrı bir önem kazanmaktadır. Standart bir akıllı kart üzerinde Ram Bellek, Rom Bellek, merkezi işlem birimi ve elektronik silinebilir bellek bulunmaktadır [2]. Bu özellikleri sayesinde birçok alanda güvenliği ve otomasyonu sağlayacak uygulamalar akıllı kartlar ile geliştirilebilmektedir. Ancak bu kadar yaygın kullanımı ve gelişmiş özellikleri bulunmasına rağmen akıllı kartlar için yazılım geliştirmek, standart programlamadan daha karmaşık bir yapı ile uğraşmaya neden olmaktadır. Ayrıca az sayıda kişi, akıllı kartlar için yazılım geliştirme sürecinde aktif rol oynamaktadır.

Bu çalışmada, uygulama geliştirme süresince yaşanılan sıkıntıları ve akıllı kartlara özel programlama dili bağımlılığını ortadan kaldırmak amacıyla model güdümlü bir yazılım geliştirme yöntemi tanıtılmaktadır. ISO/IEC 7816 standartları ve akıllı kart uygulaması geliştirirken gereken ortak programlama özellikleri gözönünde bulundurularak genel bir yazılım metamodeli ve model dönüşümlerine dayanan bir akıllı kart uygulama geliştirme sistemi önerilmektedir. Bu sistem içinde metamodelin geliştiriciler için kolay kullanılabilmesi açısından grafiksel bir arayüz tasarlanarak, bu sayede örnek modeller üretilebilmesi sağlanmıştır. Akıllı kart metamodelinden üretilen örnek modeller, hazırladığımız dönüşüm kodları ile popüler akıllı kart platformlarından olan Java Card

[3] ve Basic Card [4] modellerine dönüştürülmekte ve sonrasında bu modellerden otomatik kod üretimi sağlanmaktadır. Java Card ve Basic Card seçimi, akıllı kart sektöründeki yaygınlıkları ve programlama geliştirme ortamları dikkate alınarak yapılmıştır.

Çalısmanın dayandığı yaklaşım ikinci bölümde anlatılmaktadır. Üçüncü bölümde, akıllı kart uygulamaları için geliştirilen modeller, platform bağımsız akıllı kart modelinden Java Card ve Basic Card modellerine dönüşümler ve örnek bir elektronik cüzdan uygulaması tanıtılmaktadır. Dördüncü bölümde, elektronik cüzdan uygulamasının modellenmesi ve otomatik kod üretme kısmı anlatılmaktadır.

2 Yaklaşım

Model Güdümlü Geliştirme (MDD – Model Driven Development) ile yazılım geliştirme süreci modeller üzerinden tanımlanmaya çalışılmıştır. Kullanılan modeller ile yazılım geliştirme sürecine soyut bir bakış açısı getirilmiş ve farklı yazılım geliştirme platformlarının karmaşıklığı azaltılmaya çalışılmıştır. Object Management Group (OMG) tarafından MDD kapsamında önerilen Model Tabanlı Mimari [5] ile geliştirilmesi hedeflenen yazılım sistemlerinin farklı soyutlama seviyelerine ait modelleri ve bu modeller arasında model dönüşümleri tanımlanarak farklı uygulama platformları için bu yazılımların hızlı ve etkin bir biçimde geliştirilmesi hedeflenmektedir. Bu amaçla akıllı kart tiplerinin yapıları incelenip, yazılım geliştiriciler için otomatik kod üretme yöntemleri sağlanabilmesi ve bu yöntemler ile platform bağımsız olarak çalışılabilmesi düşünülmektedir.

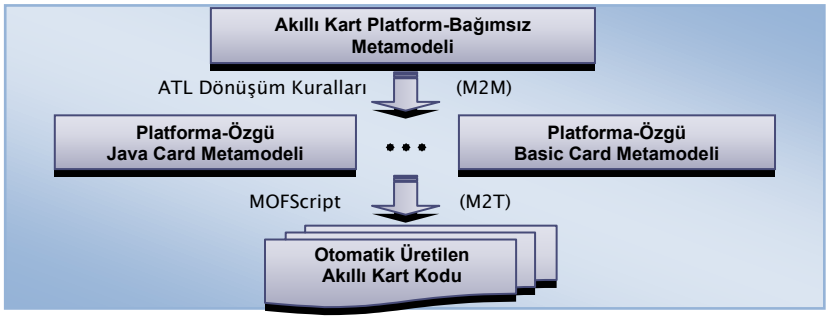
Çalışma kapsamında, akıllı kartlar için ortak özellikler barındıran ve platformdan bağımsız bir şekilde uygulama geliştirebilmeyi sağlayacak bir metamodel ve sistem geliştirilmiştir. Bu sayede geliştiricilerin dile bağlı özelliklerden kurtulup, akıllı kartlar ile ilgili genel özellikleri, ISO/IEC 7816 standartlarını bilmeleri ve programlama mantığını anlamaları yeterli olmaktadır.

Akıllı kart metamodeli Eclipse Modelleme Ortamı (Eclipse Modeling Framework - EMF) üzerinde Ecore metamodeli ile tasarlanmıştır [6]. EMF, yeni bir model geliştirebilme ve bu model için gerçek zamanlı kod üretebilmeyi sağlayan araçlar sunmaktadır. EMF platformunun sağladığı araçlar ile hazırladığımız akıllı kart metamodeli, akıllı kart yazılımları geliştirebilmek için gereken birçok model elamanını içinde barındıran platform bağımsız bir metamodeldir (platform independent metamodel – PIMM). Ecore gösterimi, tasarlanan akıllı kart metamodeli kullanılarak üretilen örnek modellerin doğru ilişkiler ile kurulabilmesini sağlamakta ve platforma özgü modellere dönüşümler kontrollü bir şekilde yapılabilmektedir.

Akıllı kart metamodelinin kullanılabilirliğini artırmak amacıyla Eclipse Graphical Modeling Framework'e (GMF) [7] dayalı grafiksel bir araç tasarlanmıştır. GMF, kullanıcılar için, EMF üzerine dayalı kullanışlı araçlar ve gerçek zamanlı bir grafiksel arayüz geliştirme ortamı sunmaktadır. Kullanıcılar zengin içerikli grafiksel arayüzler içeren GMF ortamını kullanarak kendi modellerini geliştirebilmektedirler.

Akıllı kart metamodeli ile oluşturulan örnek modeller, Atlas Transformation Language (ATL) [8] ile hazırlanan modelden modele (M2M) dönüşüm kuralları uygulanarak platforma (örneğin JavaCard ve BasicCard) özgü modellere (Platform Specific Model – PSM) dönüştürülmektedir. ATL dönüşümleri sonucunda hazırlanan modeller, grafiksel olarak platforma özgü modeller üzerinde tekrar oluşturulabilmekte ve bu örnek modellerden kod üretme aşamasına geçilebilmektedir.

Akıllı kart grafiksel arayüzü ile oluşturulacak modellerden kod üretme (M2T) aşaması MOFScript[9] ile yapılmaktadır. MOFScript, Eclipse üzerinde kulanılabilen ve modelden kod üretmeye yarayan bir araçtır. Üretilen kodlar yüksek oranda doğrudan derlenebilir ve küçük değişiklikler ile çalıştırılabilir durumda olmaktadır. Şekil 1'de akıllı kartlar için tasarlanan modelgüdümlü yazılım geliştirme yaklaşımı gösterilmiştir.



Şekil 1. Akıllı kart yazılımlarının model güdümlü geliştirilmesi yaklaşımı

3 Akıllı Kartlar

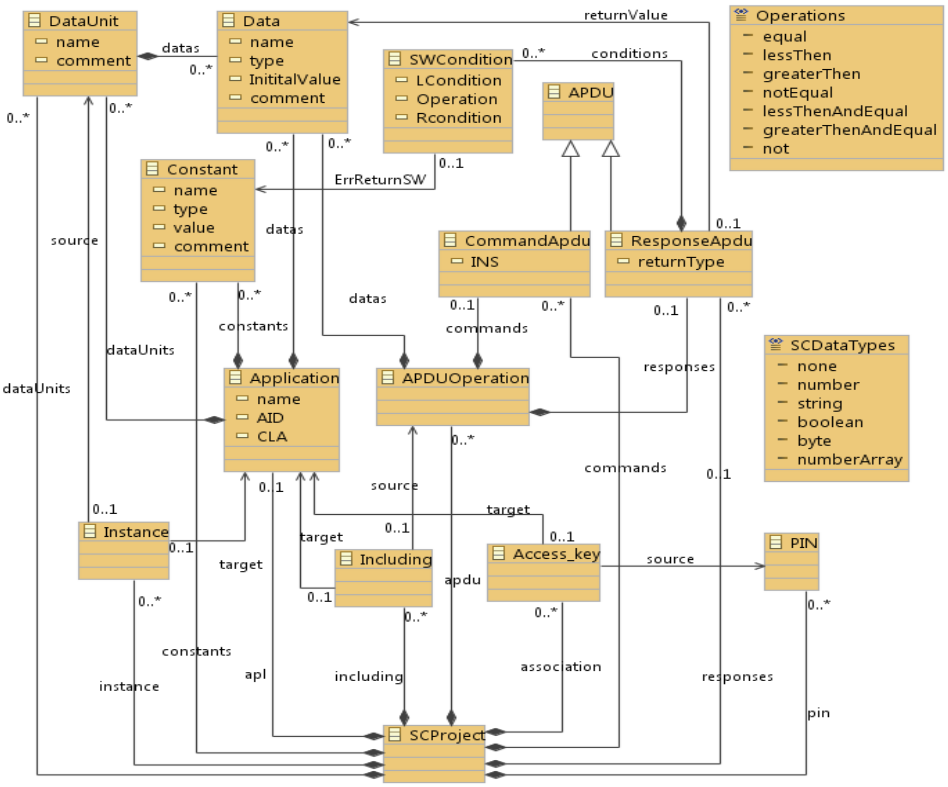
ISO/IEC 7816 standardı ile akıllı kartlar için karakteristik ve temel fonksiyonel özellikler tanımlanmaktadır. Bu sayede akıllı kartlar için standart bir iletişim alt yapısı geliştirilmiştir. Akıllı kartlar ile okuyucu arasında tüm iletişim APDU (Application Protocol Data Unit – Uygulama Protokolü Veri Yapıları) paketleri ile sağlanmaktadır. Terminal programları akıllı karta komut APDU (command APDU) paketleri göndererek karşılığında cevap APDU (response APDU) paketi beklerler. Gelen komut APDU paketine göre, akıllı kart işletim sistemi tarafından daha önceden kart içine yüklenmiş uygulama seçilir. Seçilen akıllı kart uygulaması, gelen APDU paketini işleyerek cevap APDU paketini terminal uygulamasına gönderir.

3.1 Platform-Bağımsız Akıllı Kart Metamodeli

Akıllı kartlar için haberleşme alt yapısı standartlaşmış olsa da, akıllı kart içine yüklenen uygulamalar farklı programlama dillerinde geliştirilmektedir. Bu çalışmada kod üretme amacıyla hedef olarak belirlediğimiz akıllı kartlar için de farklı platformlar üzerinde yazılım geliştirilmektedir. Java Card için Java Card Geliştirme Ortamı (Java Card Development Environment) ve Basic Card için ZC-Basic dili (ZeitControl-Basic Language) kullanılmaktadır.

Akıllı kart metamodeli, farklı akıllı kart platformlarını tek bir çatı altında toplayabilmek için tasarlanmış bir platform bağımsız modeli (PIM) tanımlamaktadır. Bu çalışmada

yalnızca Java Card ve Basic Card platforma özgü modelleri tasarlanmış olsa da, akıllı kart metamodeli daha sonra eklenebilecek platforma özgü modellere de uyum sağlayabilecek şekilde geliştirilmeye çalışılmıştır. Eclipse modelleme ortamı kullanılarak Ecore formatında hazırlanan akıllı kart metamodeli Şekil 2'de gösterilmektedir. Kolay okunabilirliğinin olması ve yer kısıtlarından dolayı metamodel elemanlarının bazı özellikleri bu resim üzerinde gösterilmemektedir.



Şekil 2. Akıllı Kart Metamodeli

Application model elemanı metamodelin anahtar öğesidir. Esas olarak bir akıllı kart programını temsil etmektedir. Application elemanı, akıllı kart uygulaması geliştirirken bir başlangıç noktası olarak görülmekte ve her akıllı kart örnek modeli içine eklenmektedir. Java Card metamodelindeki *Applet* elemanı ve Basic Card metamodelindeki *File Definition* elemanı, *Application* model elemanına karşılık gelmektedir.

Akıllı kart metamodelindeki bir diğer önemli eleman *APDU* öğesidir. Akıllı kart standartlarında da belirtildiği üzere kart ile terminal program arasındaki haberleşme *APDU* paketleri ile yapılmaktadır. Benzer şekilde bu işlemleri karşılamak amacıyla *APDU* model elemanı oluşturulmuştur. *APDU* model elemanı *commandAPDU* ve *responseAPDU* model elemanları için bir meta eleman olarak tanımlanmıştır. Bu

şekilde tasarlanan *APDU* modeli ile uygulama geliştirme aşamasının anlaşılır olması ve platforma özgü modellere dönüşümlerin kolaylaştırılması amaçlanmıştır.

commandAPDU model elemanı için belirtilen *INS* (Instruction Byte(s)) öğesi ile, program içinde çalıştırılacak komutların kontrolü sağlanmaktdır. Girilen *INS* değerine göre gelen *APDU* paketleri işlenerek istenilen *commandAPDU* öğresi çalıştırılmaktadır. Gelen *APDU* paketine bir cevap döneceği durumlarda veya cevap hazırlama aşaması işlemleri için *responseAPDU* elemanı kullanılmaktadır. Gelen APDU paketinin işlenmesi sırasında, *responseAPDU* tarafından durum komutları (Status Words-SW) gönderilmektedir. Herhangi bir koşul oluşması durumunda gönderilecek SW değerleri için *SWCondition* model elemanı tanımlanmıştır. *responseAPDU* içinde tanımlanabilen *SWCondition* elemanı ile istenilmeyen bir koşul oluşması durumunda sabit olarak tanımlanmış SW değerlerinin geri dönüşü sağlanmaktadır. Bir akıllı kart uygulaması geliştirirken çalıştırılması gereken işlemler için tanımlanan *commandAPDU* ve *responseAPDU* elemanları *APDUOperation* ismi altında oluşturulan metamodel öğesi içinde tanımlanabilmektedir. Bu sayede akıllı kart *Application* elemanı ile ilişkilendirilen bir *APDUOperation* elemanı, içerisinde, *commandAPDU* ve *responseAPDU* elemanlarını içermektedir. *APDUOperation* elemanı ile, komut veya işlemler çalıştırılırken gerekecek veri tipi veya değişken tanımlaması yapılabilmektedir. *Application* elemanı ile *APDUOperation* elemanlarını ilişkilendirmek amacıyla *Including* ilişki oku tanımlanmıştır. Bu amaçla eklenen her *APDUOperation* elemanı bir *Including* ilişki oku ile *Application* öğesine bağlanmalıdır.

Geliştirilen akıllı kart uygulamalarına yetkili erişimi sağlamak amacıyla metamodel içerisinde *PIN* elemanı tanımlanmıştır. Okuyucu ile akıllı kart arasında bir bağlantı açılmadan önce *PIN* değerinin doğrulaması yapılmaktadır. Okuyucu *PIN* değerini içeren bir *APDU* paketini gönderir ve akıllı kart içindeki uygulama bu değerin kontrolünü yapar. Örnek model oluşturulması sırasında eklenen *PIN* elemanı ile, doğrulama ve PIN değerini ilkleme komutları otomatik kod üretme aşaması çalıştırıldıktan sonra üretilen koda eklenmektedir. Doğrulama yapılmadığı sürece diğer komutlar çalıştırılamayacaktır. Akıllı kart modeline *PIN* elemanı ekleyebilmek amacıyla *Access_key* ilişkisi tanımlanmıştır. Geliştirilen uygulama içerisine *PIN* özelliklerini eklemek için *Application* ve *PIN* elemanları arasında *Access_key* ilişkisi tanımlanmalıdır.

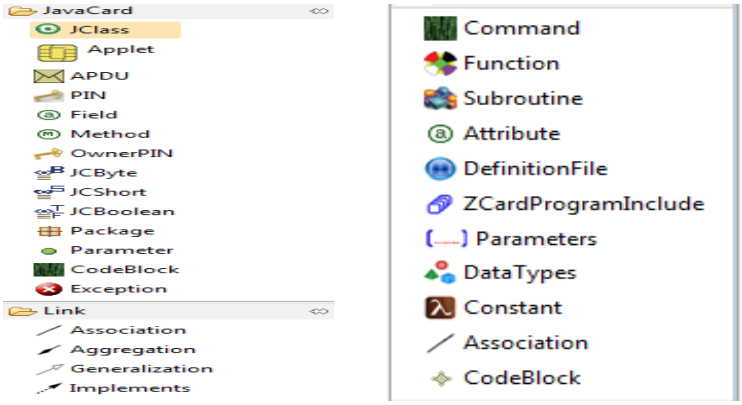
Constant ve *Data* elemanları veri tipi tanımlamaları yapmak amacıyla kullanılmaktadır. *SCDataTypes* model elemanı sayesinde beş adet veri tipi tanımlanabilmektedir. Bunlar *number*, *numberArray*, *string*, *boolean*, *byte* veri tipleridir. Örnek model içerisinde yapılan *Constant* ve *Data* tanımlamalarına göre, tüm veri tipleri, dönüşüm yapılacak model içindeki uygun veri tiplerine çevrilmektedir. Kullanıcı tanımlı veri tipleri ise *DataUnit* elemanı ile tanımlanabilmektedir. Bu sayede farklı veri tiplerini bir araya toplayarak yeni bir veri tipi oluşturulabilmektedir. Uygulama içerisinde yeni bir veri tipi kullanabilmek amacıyla tanımlanan *Instance* ilişkisi, *Application* ve *DataUnit* elemanları arasında kurulmalıdır.

3.2 Platforma Özgü Modeller (Java Card PSM – Basic Card PSM)

Platform bağımsız seviyede akıllı kart metamodeli ile modellenebilen uygulamalarda platforma özgü seviyede değişiklikler yapabilmek için, Java Card ve Basic Card

metamodelleri tanımlanmıştır. Java Card ve Basic Card için geliştirilen metamodeller ve grafiksel modelleme kullanıcı arayüzü ile akıllı kart uygulamalarına dile özgü seviyede de müdahale edilebilmesi sağlanmıştır.

Java Card ve Basic Card için geliştiricilerin uygulama modelleyebilmelerini sağlayan metamodel elemanlarının palet görüntüsü Eclipse GMF ortamında Şekil 3'teki gibi tanımlanmıştır. Java Card için önerdiğimiz platforma özgü metamodel, bu metamodele dayanan örnek modellerde kullanılan elemanlar ve Java Card akıllı kartları için kod üretme aşaması ile ilgili bilgi [10]'da ayrıntılı olarak verilmektedir. Bu sebeple platforma özgü metamodel açıklamaları bu bildiride sınırlı tutulmuştur.



Şekil 3. Java Card ve Basic Card Modelleme Elemanları

Java Card metamodelinin anahtar öğesi Applet model elemanıdır. Applet, akıllı kart tarafında bulunan Java Card programını temsil etmektedir. Java Card programında *process* metodu tarafından işlenen APDU paketlerini kullanacak işlemler için *Method* ve *APDU* elemanları tanımlanmıştır. OwnerPIN ve PIN, Java Card programında kullanılabilecek şifre işlemleri için tanımlanmış model elemanlarıdır. Bunların dışında Java Card diline özgü veri tipi tanımlamaların yapılabildiği, sınıflar arasında ilişkilerin kurulabildiği elemanlarda palet üzerinde yer almaktadır.

Basic Card metamodelinde bulunan *ZCardProgram* elemanı Basic Card için bir kaynak programı temsil etmektedir. Uygulama içerisinde prosedür tanımları *Command*, *Subroutine* ve *Function* elemanları ile yapılmaktadır. Bu elemanların Basic Card uygulamasında ayrı ayrı kullanıldıkları yerler ve çeşitli özellikleri bulunmaktadır. Tanımlanabilecek veri tipleri, sabitler ve elemanlar arasında kurulabilecek ilişkiler palet üzerinde görülmektedir. Hem Java Card hem de Basic Card modelleme ortamında geliştiricilerin o anda aklına gelebilecek kod ayrıntılarını yazabilecekleri CodeBlock elemanı tanımlanmıştır. Bu alana eklenecek kod blokları, üretilen kod içerisine kullanıcı tanımlı alanlar olarak eklenecektir.

3.3 Akıllı Kart Uygulamaları için Modelden Modele Dönüşüm (M2M)

Akıllı kart modelleme ortamı üzerinde oluşturulan örnek modeller ATL dönüşüm kuralları ile platforma özgü örnek modellere dönüştürülmektedir. Grafiksel modelleme arayüzü ile oluşturulan örnek modeller için otomatik olarak oluşturulan Ecore formatı, ATL dönüşüm kurallarına girdi olarak verilmektedir. Akıllı kart metamodelinden

üretilen örnek modellerde bulunan elemanların dönüşüm sonucunda Java Card ve Basic Card modellerinde karşılık geldikleri elemanların bir kısmı Tablo 1'de verilmiştir.

Tablo 1. Akıllı Kart Model elemanları ve çeşitli PIM elemanları arasındaki eşleşmeler

Akıllı Kart Metamodeli Platform Bağımsız Metamodeli	Java Card Metamodeli	Basic Card Metamodeli
SCProject	JCProject	ZCardProgram
Application	Applet	DefinitionFile
APDUOperation (command – response)	Method	Command
PIN	PIN	Attribute
Constant	Field	Constant

Yer kısıtları dolayısıyla tüm eleman dönüşümlerinin verilemediği tabloda APDUOperation elemanı ile ilgili birkaç ayrıntıdan bahsetmek gerekir. Bu elemandan platforma özgü modele dönüşüm ile oluşturulacak *Method* ve *Command* elemanlarının tipi, geri dönüş değeri, aldığı parametre ve içlerinde tanımlanan veri tipleri belirlenebilmektedir. Bunun dışında *Constant* elemanının da, karşılık geldiği metamodellerdeki elemanların kısıtları dikkate alınarak dönüşümü yapılmaktadır. Model dönüşümleri için geliştirilen ATL kurallarının bazıları Tablo 2'de gösterilmektedir.

Tablo 2. Platforma özgü modellere dönüşüm için yazılan bazı ATL kuralları

SmartCard2JavaCard.atl

```
rule SmartCard2JavaCard{
    from
        smartCard : MM!SCProject
    to
        javaCard : MM1!JCProject(
          title <- smartCard.title,
          rootApplet <- Set{MM !
Application.allInstances()},
          ownerPin <-
Set{MM!PIN.allInstances()},
           aggregation <-
Set{MM!Access_key.allInstances()})}
```

```
helper context MM ! Application def :
getAssociations():
    MM ! Application = MM !
Including.allInstances()
    -> select(includes | includes.target
= self);
```

```
rule Application2Applet{
    from
        appl : MM!Application
    to
        app_let : MM1 ! Applet(
         name    <- appl.name,
         fields<-
Sequence{appl.constants, appl.datas},
         methods<-
appl.getAssociations())}
```

SmartCard2BasicCard.atl

```
rule SmartCard2BasicCard{
    from
        smartCard : MM!SCProject
    to
        javaCard : MM1!ZCardProgram(
       name <- smartCard.title,
        commands <- Set{MM !
APDUOperation.allInstances()->
select(a|a.oclIsKindOf(MM ! CommandApdu)
= false)},
       attributes<-
Set{MM!PIN.allInstances()->
select(a|a.oclIsKindOf(MM !
APDUOperation) = false)},
       defFile     <-
Set{MM!Application.allInstances()})}
```

```
rule Application2DEF {
        from
         cons : MM!Application
        to
         defs : MM1!DefinitionFile (
name <- cons.name,
constants<- Sequence{cons.constants}
)}
```

```
rule Constant2Constant {
        from
         cons : MM!Constant
        to
         defs : MM1!Constant (
        name <- cons.name,
        InitialValue     <- cons.value
)}
```

Bu kurallar ile oluşturulan paltforma özgü örnek modeller tekrar eklenti yapmak veya düzeltmek amacıyla grafiksel ortamda açılabilmektedir. Görsel modelleme ortamı üzerinde dile özgü eklentiler yapılabilmekte ve üretilecek kod oranı artırılabilmektedir.

3.4 Akıllı Kartlar için Örnek Cüzdan Uygulaması

Geliştirilen akıllı kart modelinin özelliklerini ve kısıtlarını örnek üzerinde anlatabilmek amacıyla bir elektronik cüzdan uygulaması ile modelin kullanımı gösterilmeye çalışılmıştır. Akıllı kartlarda geliştirilebilen cüzdan uygulaması ile kullanıcıların yanlarında taşıyabileceği küçük boyuttaki kartları kullanması sağlanarak toplu ulaşım, otopark, sinema, telekomünikasyon gibi alanlarda fiziksel olarak taşınan para trafiği azaltılabilmektedir. Yüksek güvenlik seviyesi, kartlar üzerinde grafiksel ve yazılımsal olarak kullanıcı kişiselleştirmesi yapılabilmesi ve takip edilebilmesinin kolay olması gibi nedenlerden dolayı akıllı kartlar ve bu kartlar üzerinde geliştirilen elektronik cüzdan uygulamaları gibi projeler, ticari ve bireysel anlamda daha birçok konuya kaynak olabilmektedir. Çalışmamızda geliştirdiğimiz model ortamını temel özellikleri ile tanıtabilmek ve program zorluğunu azaltmak amacıyla seçilen akıllı kart cüzdan uygulamasının basit ve anlaşılır olması düşünülmüştür.

4 Akıllı Kart Uygulamalarının Grafiksel Modellenmesi

Üçüncü bölümde ayrıntıları anlatılan akıllı kart metamodeli ile Eclipse GMF üzerinde akıllı kart uygulamaları geliştirmek amacıyla grafiksel bir arayüz tanımlanmıştır. Modelleme ortamı ve metamodel öğeleri görsel olarak Şekil 4 üzerinde gösterilmektedir. Akıllı kart uygulamasına eklenebilecek metamodel elemanları modelleme ortamının sağ tarafında görülmektedir. Metamodel elemanları arasında kurulabilecek ilişki tipleri için, aynı alan üzerinde *Link* sekmesi bulunmaktadır. Yeni oluşturulan elemanlar ve ilişki tipleri arasındaki kurallar modelleme ortamı tarafından denetlenmektedir. Bu sayede elemanlar ve ilişkiler arasındaki yanlış tanımlamalar editör üzerinde engellenmektedir.

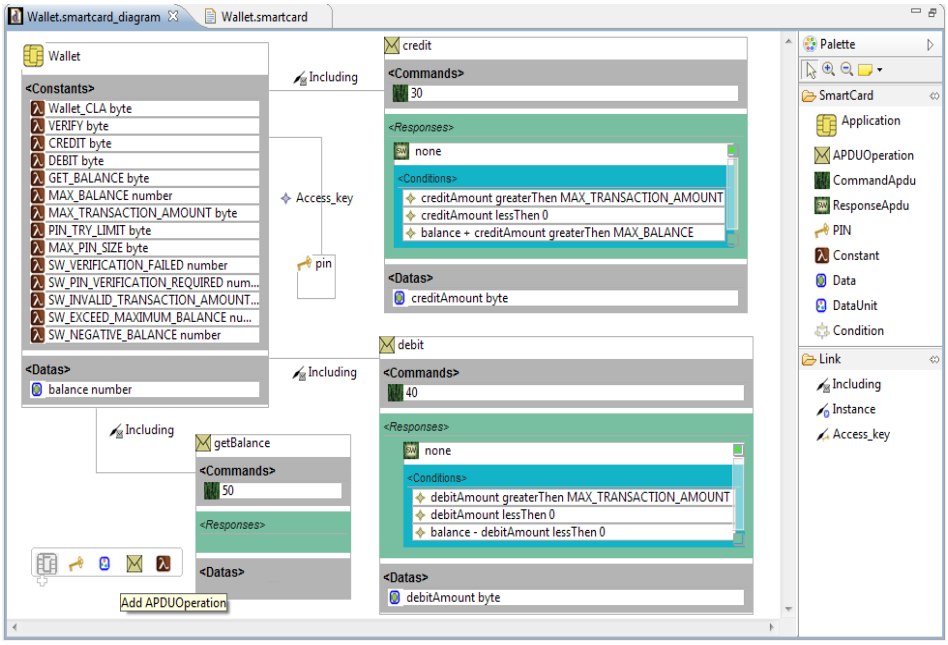
Bu kontrollerden bazıları şu şekilde tanımlanmıştır: *Command* ve *response APDU* elemanları sadece *APDUOperation* elemanı içerisinde tanımlanabilmektedir. İlişki okları sadece ilgili elemanlar arasında oluşturulabilmektedir. *Constant* elemanı *Application* içinde ya da *DataUnit* içinde tanımlanabilir. *Constant* ve *Data* elemanları ilgili olmadıkları bir alanda kullanılamazlar. Bir adet *Application* elemanı tanımlanabilir.

4.2 Akıllı Kart Uygulamaları için Modelden Kod Üretme

Akıllı kart örnek modelinden, platforma özgü modellere Bölüm 3.3'te anlatılan dönüşümler ile oluşan hedef modellerden kod üretmek amacıyla MofScript dili kullanılarak M2T kodları hazırlanmıştır. MOFScript, Eclipse içinde bir araç olarak kullanılabilmekte ve geliştirilen modellerden herhangi bir anda kod üretilebilmektedir. Platforma özgü Java Card modelinden kod üretilmesi ve akıllı kart koduna dönüşüm için yazılan bazı MOFScript kodları [10]'da ayrıntılı olarak anlatılmıştır.

Grafiksel arayüz üzerinde tasarlanan modellerin çıktısı, MOFScript dili için bir girdi olarak algılanmakta ve Java Card ya da Basic Card için yazılan modelden koda dönüşüm kodları ile akıllı kart programı elde edilmektedir. Modellemesi yapılan cüzdan

uygulamasının Java Card platformuna özgü modeline dönüşüm işlemleri tamamlandıktan sonra, MOFScript kodlarının çalıştırılması ile elde edilen koddan örnek bir bölüm Şekil 5'te gösterilmektedir.



Şekil 4. Platform-Bağımsız Akıllı kart modelleme araçları ile tasarlanan Cüzdan Uygulaması

```
package bank.purse;

  import javacard.framework.*;
  import javacardx.framework.*;

  public class Wallet extends Applet {
    public  static  final byte Wallet_CLA = 0xB0; // code of CLA byte in the command APDU header
    public  static  final byte VERIFY = 0x20; //
    public  static  final byte CREDIT = 0x30; //
    public  static  final byte DEBIT = 0x40; //
    public  static  final byte GET_BALANCE = 0x50; //
    public  static  final short MAX_BALANCE = 0x7FFF; // maximum balance
    public  static  final byte MAX_TRANSACTION_AMOUNT = 127; //    maximum transaction amount
    public  static  final byte PIN_TRY_LIMIT = 0x03; // maximum number of incorrect tries
    public  static  final byte MAX_PIN_SIZE = 0x08; //
    public  static  final short SW_VERIFICATION_FAILED = 0x6300; //
    public  static  final short SW_PIN_VERIFICATION_REQUIRED = 0x6301; //
    public  static  final short SW_INVALID_TRANSACTION_AMOUNT = 0x6A83; //
    public  static  final short SW_EXCEED_MAXIMUM_BALANCE = 0x6A84; //
    public  static  final short SW_NEGATIVE_BALANCE = 0x6A85; //
    OwnerPIN pin;
    short balance;

    public static void install(byte[] bArray, short bOffset, byte bLength) {
     new Wallet();
    }

    public Wallet() {
        pin = new OwnerPIN((byte)(5),(byte)(8));
      pin.update(/*write your pin value variable here*/, (short)(0), (byte)(/*write your size of pin here in bytes*/));
      register();
    }
```

Şekil 5. Cüzdan uygulaması için üretilen akıllı kart kodunun bir kısmı

5 İlgili Çalışmalar

Bu alanda daha önce yapılan çalışmalarda, akıllı kartlar için kod üretme, kişiselleştime ve üretim aşamalarını hızlandırmak ve kolaylaştırmak amacıyla model güdümlü yaklaşımlar kullanılmıştır. Akıllı kartlar için üretim aşamasından sonra kişiselleştirme ve ilgili uygulamaların yüklenmesi gibi işlemler yapılmaktadır. Bu üretim ve konfigürasyon aşamasını hızlandırmak ve birleştirmek amacıyla model tabanlı yaklaşım kullanan bir yöntem [11]'de verilmiştir.

Başka bir çalışmada, MetaSlang dili kullanılarak, Java Card applet kodlarının üretilmesi ve üretilen kodun doğruluğunun denetlenmesi ile ilgili bir yöntem anlatılmaktadır [12]. Bu dil, Java Card programlamaya yakın, görsel olmayan bir yapıda geliştirmeye olanak vermektedir. Geliştiricilerin bu konuda bilgili olmaları beklenmektedir.

UML diyagramları kullandırılarak, platform bağımsız bir model oluşturulup, bu modelden kod üretimi sağlayan bir yöntemde [13], akıllı kart kodlarının üretilmesi sağlanmaktadır. Bu çalışmada, Java Card dili tanımlamaları yerine UML tanımlamaları kullanılmış ve bu sayede geliştiricilerin Java Card kodu üretebileceği anlatılmıştır. Biz çalışmamızda, platform bağımsız modelde, Java Card dil öğelerine yakın model elemanları kullanarak, orta düzey geliştiriciler için yeni bir dil öğrenmelerine gerek kalmadan tasarım yapabilmelerini ve yeni başlayanlar için kolay kullanılır bir geliştirme ortamı sunmayı amaçladık. Benzer konuda yapılan birçok çalışmada, görsel olarak tasarlanabilen, tut-bırak-düzenle tarzı uygulamalar bulunmamaktadır. Bu çalışmamızda geliştiricilerin, akıllı kart programlamanın zorluklarından uzaklaşıp görsel öğeler kullanarak oluşturabilecekleri tutarlı örnek modellerden hatasız kod üretebilmelerini sağlamaya çalıştık. Bu amaçla geliştirilen tüm model üretme, üretilen modelden kod üretme aşamaları, yaygın bir editör olan Eclipse üzerinde gerçeklenebilmektedir.

6 Sonuç

Gelişen yazılım dünyasında, model tabanlı yaklaşımlar büyük önem kazanmaya başlamıştır. Alt seviye hatalar, dil bağımlılıkları ve küçük değişiklikler için bile zaman alan uğraşlarda bulunmak geliştiriciler için büyük enerji kaybına yol açmaktadır. Bu nedenle benzer ilgi alanındaki ürünler için ortak geliştirme ortamları tasarlamanın büyük faydaları olacaktır. Çalışmamızda kullandığımız yöntemin ve yaklaşımın akıllı kart yazılımlarının geliştirilme aşamasına farklı bir bakış açısı getireceğine inanmaktayız. Bu çerçevede yapılan çalışmamızın devamında otomatik üretilen kodların verimlilik testleri, gerçek koda oranla ne kadar üretilebildiğinin araştırılması ve geliştirilmesi yer almaktadır.

Kaynakça

1. **ISO/IEC 7816** Standards family for Identification cards - Integrated circuit cards,http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45144
2. **Rankl, W., Effing, W.**: Smart Card Handbook. John Wiley & Sons, West Sussex (2000).
3. Sun Microsystems: Java Card Technology, http://java.sun.com/javacard/
4. ZeitControl Card Systems GmbH: Basic Card, http://www.basiccard.com/
5. Object Management Group Model Driven Architecture, http://www.omg.org/mda/
6. Eclipse Modeling Framework Project (EMF) , http://www.eclipse.org/modeling/emf/
7. Eclipse Graphical Modeling Framework (GMF) , http://www.eclipse.org/modeling/gmf/

8. **Jouault, F., Kurtev, I.**: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128--138. Springer, Heidelberg (2006)
9. **Oldevik, J., Neple, T., Gronmo, R., Aagedal, J., Berre, AJ.**: Toward Standardised Model to Text Transformations. In: Hartman A., Kreische D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 239--253. Springer, Heidelberg (2005)
10. **Sarıtaş, H. B., Kardas, G.,** Java Card Yazılımlarının Model Güdümlü Geliştirilmesi, Turkish Workshop on Model Driven Software Development, Bilkent Üniversitesi, Mayıs 22
11. **Bonnet, S., Potonniee, O., Marvie, R., Geib, J-M.**: A Model-Driven Approach for Smart Card Configuration. In: Karsai, G. Visser, E. (eds.) GPCE 2004. LNCS, vol. 3286, pp. 416--435. Springer, Heidelberg (2004)
12. **Coglio, A.**: Code generation for high-assurance Java Card applets. In: 3rd NSA Conference on High Confidence Software and Systems, pp. 85--93 (2003)
13. **Moebius, N., Stenzel, K., Grandy, H., Reif, W.**: Model-Driven Code Generation for Secure Smart Card Applications. In: 20th Australian Software Engineering Conference, pp. 44--53. IEEE Computer Society Press, New York (2009)

WEB TABANLI SİSTEMLERİN GELİŞTİRİLMESİNE YÖNELİK PLATFORMA ÖZGÜ BİR ÜSTMODEL

Ferhat Erata[1], Geylani Kardaş[2]

[1,2] Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, 35100, Bornova, İzmir
[1]ferhat.erata@unitbilisim.com, [2]geylani.kardas@ege.edu.tr,

Özet. Web uygulamalarını tasarlamak ve geliştirmek için çeşitli web mühendisliği yaklaşımları bünyesinde sistematik tasarım ve model dönüşümleri sonrasında web uygulamalarının otomatik olarak üretilmesi bulunmaktadır. Ancak kabul görmüş belirli bir platform teknolojisi olmaksızın gerçekleştirilen otomatik üretim, kurumun iş ihtiyaçlarının karşılanması açısından hem üretken hem de etkin olmamaktadır. Bu bildiride, PSM4WSS isimli platforma özgü bir web mühendisliği üstmodeli önerilmektedir. Microsoft SharePoint uygulama çatısı üzerinde geliştirilen ve Model Güdümlü Mimari'ye dayanan bir yöntem ile önerilen üstmodelin kullanıcı modellerinden web tabanlı kurumsal çözümlerin otomatik olarak üretilmesi sağlanmaktadır. Önerilen üstmodel, Windows Sharepoint hizmetleri ("*Windows SharePoint Services*") (WSS) nesne modelinin içerik yönelimli ayrık kavramlarını bakış açılarına bölerek ve bu kavramların arasındaki ilişkileri yeniden tanımlayarak soyutlama seviyesini yükseltmektedir.

1 Giriş

Web uygulamalarını model güdümlü geliştirmek için OO-H [1], UWE [2] ve WebML [3] gibi çeşitli web mühendisliği metodolojileri ve platform bağımsız üstmodelleri önerilmiştir. Bu yaklaşımların odağında modelleri, analiz ve tasarım amaçlı kullanmak dışında modellere dayalı olarak web uygulamalarının otomatik üretilmesi de bulunmaktadır. Ancak kabul görmüş ve üreticisi tarafından gelişimi sürekli desteklenen belirli bir uygulama çatısı ya da platform teknolojisi olmaksızın otomatik üretim, kurumun iş ihtiyaçlarının karşılanması ve geliştirilen uygulamaların çevikliği –değişen ihtiyaçlara hızlı bir şekilde uyum sağlayabilmesi– açısından hem üretken hem de etkin olamamaktadır. Diğer yandan platform bağımlı teknolojiler ise UWE ve benzeri üstmodellerden kendine model eşleme ve dönüşümlerinin basit bir şekilde gerçekleştirilebilmesi adına yeterince soyut değillerdir. Bunun belirgin sebebi olarak platform bağımlı üstmodellerin uygulamadaki ihtiyaçlardan, gelişen teknolojilerden gelmesi ve doğası gereği hedef bir platform teknolojisi içermesi; platform bağımsız üstmodellerin ise akademik olarak çalışılması ve genelleştirme çabası sebebiyle bu iki yaklaşım birbirlerinden bağımsız olarak gelişmektedir. Bu sebeple çalışmamızda, bu iki yaklaşım köprülenerek, Microsoft Windows platformuna özgü bir teknoloji olan Sharepoint [4] uygulama çatısının Windows Sharepoint Services (WSS) nesne modelini genişleten ve soyutlama seviyesini yükselten, web tabanlı kurumsal çözümler geliştirmeye yönelik bir yazılım ürün hattı oluşturabilecek PSM4WSS (*"Platform Specific Metamodel for Windows SharePoint Services")* ismini verdiğimiz bir web mühendisliği üstmodeli önerilmektedir. Önerilen üstmodel, Windows Sharepoint

hizmetleri nesne modelinin içerik yönelimli ayrık kavramlarını bakış açılarına bölüp bu kavramların arasındaki ilişkileri yeniden tanımlayarak soyutlama seviyesini yükseltmektedir. Web mühendisliği alanında araştırılmış uygun bir platform bağımsız üstmodeli SharePoint platformuna özgü teknoloji spesifik ilgileri kapsayacak şekilde genişletmek ya da Microsoft'un önerdiği WSS nesne modelinin soyutlama seviyesini yükseltip web mühendisliği alanında kabul görmüş ortak görünümlere parçalayıp yeniden tümleştirmek gibi iki alternatifin birleşimi üzerinden bir çözümleme ile tamamen yazılım geliştirme ihtiyaçlarına dayalı yeni bir üstmodel önerilmiştir. "*Object Management Group*"'un (OMG) önerdiği Model Güdümlü Mimari'ye (MGM) [5] dayanan bir yöntem ile PSM4WSS modellerinden web tabanlı kurumsal çözümlerin otomatik olarak üretilmesi sağlanmaktadır.

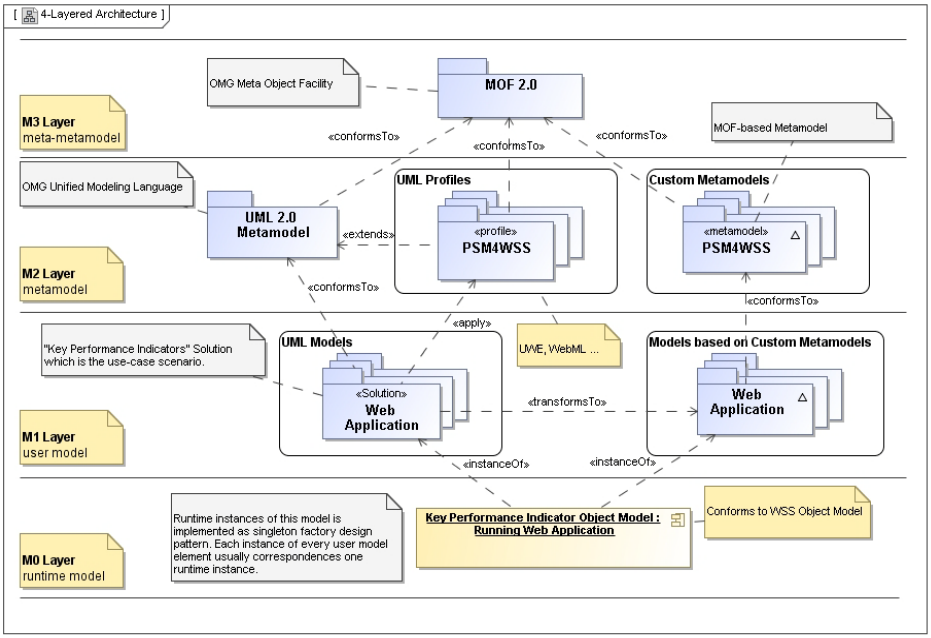
Hem kullanıcı hem de teknolojik ihtiyaçların sürekli gelişim içerisinde olması, Web sistemlerinin tasarımının ve gerçekleştiriminin de devamlı bir gelişim içerisinde olmasını güdümler. Geliştirme sürecinin her hangi bir aşamasındaki modellerin kolaylıkla bu gibi değişikliklere uyum sağlaması için gereken esnekliği yakalamada PSM4WSS, geliştirilmesinin ilk fazından itibaren katı bir şekilde ilgilerin ayrımı ("*separation of concerns*") prensibine dayanır. Farklı bakış açılarındaki modelleri ve model dönüşümlerini temel alan model güdümlü bir geliştirme süreci gerçekler. Esas mücadele ise yazılımın geliştirme maliyetlerini azaltan ve verimliliği arttıran bir yazılım ürün hattı ("*software product line*") oluşturabilecek bileşenleri, bu bileşenlerin üstmodellerinin soyutlama seviyesini yükseltip web sistemlerin spesifik bir platformda tamamen otomatik olarak gerçekleştirimine uygun bir geliştirme süreci destekleyebilmektir. Model güdümlü bir yaklaşımı takip ederek söz konusu ihtiyaçların karşılanması için çalışmamızda izlediğimiz yöntem: (a) yeni modeller ve model elemanları tanımlayarak yeni ihtiyaçların karşılanması, (b) WSS nesne modelinin bu yeni ek ihtiyaçlar ışığında tekrar tanımlanması, (c) tanımlanan yeni üstmodeli grafik bazlı modelleyebilecek bir araç geliştirilmesi, (d) üstmodelleme ve model dönüşümleri içeren bir yazılım geliştirme süreci ile birlikte PSM4WSS modellerini girdi olarak kullanan ve SharePoint platformu üzerinde otomatik bir şekilde web uygulaması üretebilmeyi olanaklı kılacak bir çatı geliştirilmesidir.

Bildirinin geriye kalan kısmı şu şekilde düzenlenmiştir. Bölüm 2'de Model Güdümlü Geliştirmeye (MGG) genel bir bakış, PSM4WSS'in MGM içerisine nasıl oturtulduğu ve PSM4WSS'in türetilmesinde kullanılan modelleme yaklaşımı anlatılmaktadır. Bölüm 3'te MOF-tabanlı üstmodel ve UML profili anlatılmıştır. Bölüm 4'te ise örnek bir senaryoya *içerik* bakış açısı ("*viewpoint*") üzerinden UML profili uygulanmıştır. Bölüm 5'te ilgili diğer çalışmalar aktarılmıştır. Bölüm 6'da ise sonuç ve ileriye yönelik çalışmalar yer almaktadır.

2 Model Güdümlü Geliştirme ve Modelleme Yaklaşımı

MGG farklı soyutlama seviyelerindeki modelleri kullanarak yazılım geliştirmedeki karmaşıklığı azaltmayı hedeflemektedir. MGG çalışma alanına özgü üst modellerin tanımlanmasına, bu üst modellere uyan sistem modellerinin oluşturulmasına, modellerin içerdiği varlıklar arasındaki eşlemelere dayalı olarak modeller arasında dönüşümlerin

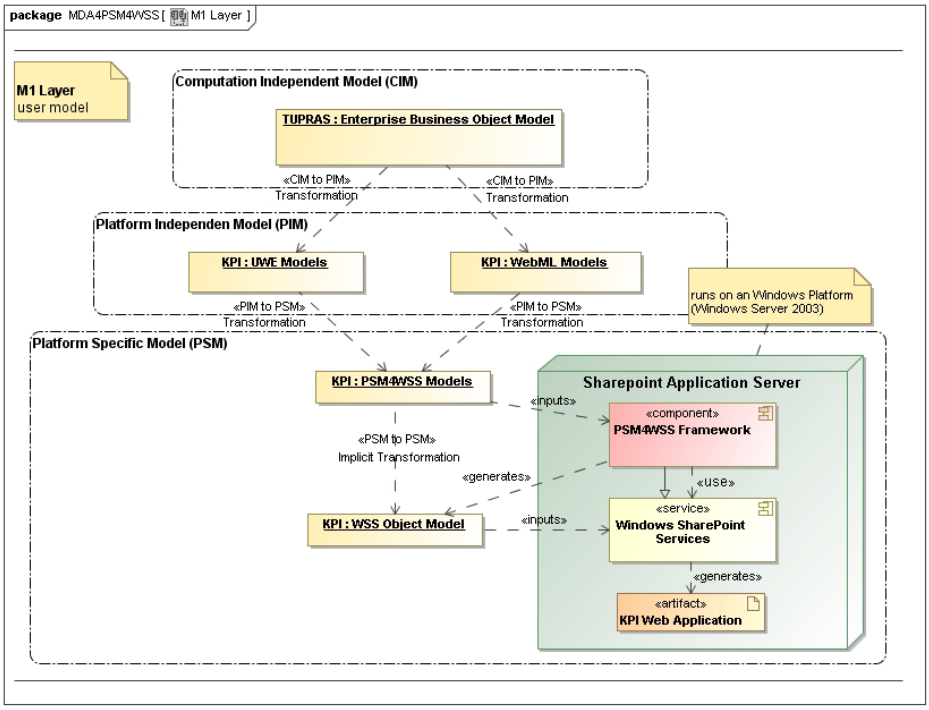
tanımlanmasına ve uygulanmasına ve son olarak çıktı modellerinden sistem yazılım kodlarının otomatik olarak elde edilmesini sağlayan modelden metne dönüşümlerin tanımlanmasına ve uygulanmasına ihtiyaç duymaktadır. Bildiride ortaya konan çalışmamız kurumsal web sistemlerinin geliştirilmesi için model güdümlü yaklaşımın tüm bu ihtiyaçlarını karşılayan bütünleşik bir yazılım geliştirme süreci sunma amacındadır. WSS üzerinde çalışacak kurumsal web uygulamalarının model güdümlü geliştirme süreci ortaya konurken OMG'nin MGG yaklaşımının bir gerçekleştirimi olarak önerdiği MGM [5] kullanılmıştır. MGM'de modeller geliştirim sürecine model dönüşüm zincirlerinden yazılım kodlarının üretilmesine kadar her safhada entegre olmuş durumdadırlar. Bu entegrasyonu sağlamak amacıyla MGM, modellerin MOF [6] temelli bir dil ile ifade edilmesine ihtiyaç duymaktadır. Şekil 1'de MGM'nin dört katmanlı temel yapısının çalışmamızdaki izdüşümü görülmektedir.



Şekil 1. OMG' nin 4-katmanlı MGM'sinin çalışmamızdaki iz düşümü

Katmanlı mimarinin en üst seviyesinde (M3) üst-üstmodel ("*meta-metamodel*") yer almaktadır. Üst-üstmodel, üstmodellerin ("*metamodel*") tanımlanması için bir zemin oluşturmaktadır. MOF standardı üst üstmodeldeki yapıları tanımlamaktadır. Mimarinin M2 seviyesinde M3'e dayalı olarak tanımlanan üstmodeller yer almaktadır. Bu seviyedeki üstmodeller model güdümlü geliştirim aracının uygulama alanına ait modellerin oluşturulabilmesi için şablon sağlamaktadır. PSM4WSS için MGM'nin M2 katmanında hem bir MOF tabanlı üstmodel hem de o üstmodelden "*Unified Modeling Language*" (UML) üstsınıflarına eşleme yolu ile bir UML profili geliştirilmiştir. M1 seviyesinde M2'de yer alan üstmodellerle tanımlanabilecek model örnekleri yer almaktadır. PSM4WSS kullanıcı modelleri bu kısımda yer alır. En alt seviye olan M0'da ise modellerin çeşitli platformlar üzerinde çalışan örnekleri bulunmaktadır.

Çalışmamızda bu katmanı oluşturan örnekler ise WSS nesne modelinde çalışan .NET platformu nesneleridir. MGM kapsamında model güdümlü geliştirme yapabilmek için öncelikle M2 seviyesindeki üstmodellerin tanımlanması gerekmektedir. MGM bu seviye için üç adet üstmodel tanımlamıştır: Yazılım Bağımsız Üstmodel ("*Computation Independent Metamodel*") (YBÜ), Platform Bağımsız Üstmodel ("*Platform Independent Metamodel*") (PBÜ) ve Platforma Özgü Üstmodel ("*Platform Specific Metamodel*") (PÖÜ). Şekil 2'de bu üstmodellerin M1 seviyesindeki model örnekleri üzerinden yerleşimi gösterilmiştir. Buna göre genel bir kapsama sahip olan örneğin UWE [2] ve WebML [3] üstmodelleri birer PBÜ'dür ve bu üstmodellere uyan model örnekleri platform bağımsız seviyede yer alır. PSM4WSS ise aynı zamanda gerçek platform uygulaması detaylarını da barındırdığından bir PÖÜ'dür ve bu üstmodele uyan örnekler WSS platformuna özgü seviyededir.



Şekil 2. M1 katmanına kullanıcı modellerinin yerleşimi

3 MOF-tabanlı Üstmodel ve UML Profili

Çalışmamız kapsamında geliştirilen PSM4WSS hem MOF-tabanlı bir üstmodel hem de UML profilidir. Önce web mühendisliği ve WSS nesne modelindeki kavram ve varlıklar üzerinden hazırlanan PSM4WSS üstmodeli türetilmiştir sonra dönüşüm ve eşleme yolu ile UML profiline dönüştürülmüştür. Bu yaklaşım hem UML uyumlu hem de MOF tabanlı araç ve ortamlar arasında örnek modellerin değişimini olanaklı kılmaktadır. UML profili, PSM4WSS üstmodeli için alana özgü özel bir modelleme aracı geliştirmeksizin profilleme mekanizmasını destekleyen araçları ("*IBM Rational*

Software Architect" [7] gibi) kullanarak kurumsal web uygulamalarının modellenmesini mümkün kılar. MOF'un desteklediği standartlardan birisi de M3-, M2-, M1-katmanlarında XML tabanlı üstveri değişim formatı olan "*XML Metadata Interchange*" (XMI)'dır. Böylece UML Profili ile üretilmiş M1 modelleri XMI serileştirme ile gerçekleştirilen PSM4WSS'e dayalı modelleme çatısına ilgili web uygulamasını üretmesi için girdi olarak verilir. .NET ve SharePoint platformu üzerinde geliştirilmiş platforma özgü ortamın PSM4WSS modellerini WSS Nesne Modeli'nin (WSSNM) nesnelerine dönüştürmesi sırasında bir web uygulamasının otomatik olarak üretilmesini sağlayan çalışma zamanı bileşenlerinin de gerçekleştirebilmesi için MOF tabanlı bir üstmodel bir gerekliliktir. Önerilen üstmodel ve UML profili aşağıdaki altbölümlerde ayrı ayrı anlatılmaktadır.

3.1 Üstmodelleme ve MOF-tabanlı Üstmodelin Türetilmesi

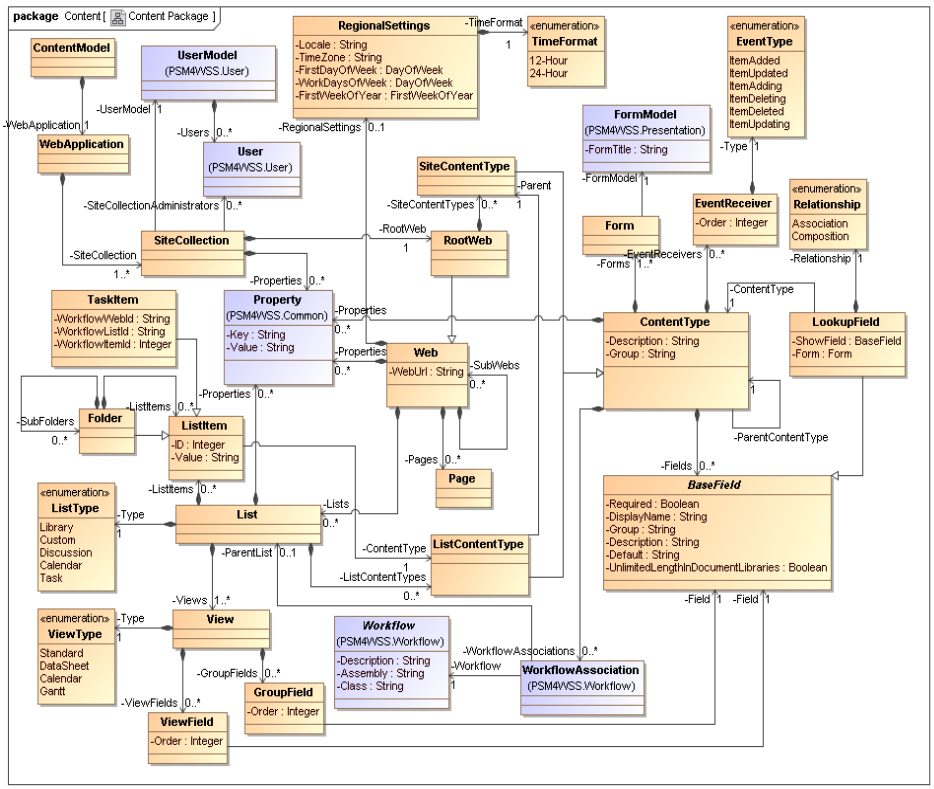
Yaygın bir pratik olarak, MOF tabanlı üstmodeller, UML gösteriminin kısıtlı bir alt kümesini kullanarak nesne yönelimli model olarak ifade edilirler. Bu yüzden PSM4WSS'in tüm paketleri bir UML sınıf diyagramı aracı ile resmedilmiştir. Üstmodelin tüm yapısı 6 ayrı paket diyagramı ile görünümlere ayrılmıştır. Her bir paket o paketin bakış açısını oluşturan bir kullanıcı modeli içerir. Örneğin içerik paketi, içerik modelini oluşturan elemanları içerir. Bir paketin içerisindeki model elemanı ile farklı paketlerdeki model elemanları arasındaki ilişkiler gösterilirken diğer paketlerdeki sınıflar farklılaştırılmıştır ve sınıf üstünde hangi paketten geldiği gösterilmiştir. Paketi gösterilmemiş ise o model elemanı anlatılan paketin bir üstvarlığıdır.

İlgilerin ayrımı ("*separation of concerns*") prensibine dayalı olarak çalışmamızda modeller, web uygulama geliştirim sürecinin ihtiyaç mühendisliği ("*requirement engineering*"), analiz, tasarım ve gerçekleştirim gibi aşamalarının değişik kademelerinde inşa edilirler ve bir web uygulamasının içerik, görünüm, gezinim yapısı ve sunum gibi değişik ilgilerine karşılık gelen farklı görünümlerini temsil etmekte kullanılırlar. PSM4WSS'i ilgilerine ("*concerns*") göre paketlere bölerek ve her pakete bir model türü atayarak, web sisteminin farklı bakış açıları ("*viewpoints*") ile geliştirilebilmesi sağlanmıştır.

PSM4WSS'in içerik modeli, uygulama alanına özgü kavramları ve kavramlar arası ilişkileri belirtmekte kullanılır. Hipermetin ya da gezinim yapısı uygulamanın içerik modelinden türetilebilmesine karşın içerikten bağımsız olarak da modellenebilmektedir. Gezinim modeli modellenen web sisteminin gezinim yollarını ifade etmekte kullanılır. Sunum modeli ise insan-makine iletişim görevlerini sağlayan web ara-yüzlerinin sunumunun temsilinde kullanılır. Yer kısıtları nedeniyle bu bildiride sadece içerik paketi resmedilmiştir (Şekil 3).

PSM4WSS, farklı görünümlerin yapısal ilgilerini belirtmek için her bir modelin görselleştirilmesinde en az bir çeşit UML diyagramı önermektedir. Öte yandan, PSM4WSS ile kurumsal web uygulama geliştirilirken web sisteminin davranışsal ilgilerini belirtmekte sıklıkla durum makinesi iş akışları üstmodeli kullanılmaktadır. Üzerinde durulan bir başka ilgi ise kişiselleştirme ya da şartlara göre web sisteminin uyum yeteneğidir. Web sistem, kullanıcıya en uygun bilgi, link veya sayfaları, şartlara bağımlı ("*context-dependent*") özelliklerin farkındalığı ile sağlayabilmektedir. Daha

uyum sağlayabilen ve kullanıcı deneyimi ("*user experience*") yüksek web sistemlerinin modellenebilmesi için üstmodel içerisinde gezinim ve kullanıcı paketlerini birleştiren bir erişim kontrolü ("*access control*") mekanizması önerilmiş ve buna uygun model elemanları üstmodele dahil edilmiştir.



Şekil 3. PSM4WSS içerik paketi

3.2 UML genişletme ve UML profilin türetilmesi

Web uygulamalarının gerçekleştirilebilmesi sırasında Sharepoint ürünlerinin ("*artifact*") yapısını ve anlamsallığını ("*semantics*") üstmodelde yakalayabilmek amacıyla çalışmalarımız kapsamında ayrıca bir takım UML genişletmeleri tanımlanmıştır.

UML'de, sterotipler ("*stereotypes*"), etiket değerler ("*tagged values*"), kısıtlar ("*constraints*") ve özel ikonları içeren standart bir genişletme mekanizması tanımlıdır. "*IBM Rational Software Architect*" [7] gibi, bir profil tanımı kabul eden ve modelleme ortamını modelleyiciyi sadece sterotip, etiket değerleri, ikonları ve kısıtlardan oluşan UML'nin küçük bir alt kümesini kullandırmaya zorlayacak şekilde biçimlendiren bir takım araçlar piyasada mevcuttur. Önerilen mekanizma sayesinde mevcut araçların UML'in özel amaçlar için özelleştirilmesi ve düzenlenmesi sağlanılarak, modelleyicinin PSM4WSS modellerini üretmesinde UML'nin ifade etme gücü yüksek somut notasyonunu kullanması sağlanmıştır.

Bir *sterotip ("stereotype")*, mevcut bir UML üstsınıfını ("*metaclass*") temel alan sanal bir üstsınıf oluşturur. Böylece temel üstsınıf örneklerini sınıflandıran bir yol sağlanmış olur ve tercihen ek kısıtlar ("c*onstraints*") ve/veya gerekli etiket değerleri ile birlikte belirtilir. Örneğin, PSM4WSS İçerik paketi model elemanı olan "*Web*" kavramı, sterotip <<web>> olarak bir UML üstsınıfı olan paketten ("*package*") genişletilir (kalıtlanır). Bir paket elemanının örneğine uygulandığı zaman örneği başka bir paket örneğinin içerisinde iç-içe geçmiş olarak göstermeyi sağlar.

Bir *etiket değeri,* isteğe bağlı bilginin bir örneğe takılmasına izin vererek bir UML üstsınıfının niteleyicisi ("*attribute*") gibi davranır. Bir dizi etiket değeri, bir sterotip ile ilişkilendirilerek sterotipin uygulandığı model elemanına takılır. Öte yandan bir *kısıt,* model elemanlarına sözel olarak belirtilebilen anlam katar. Bu belirtme, seçilmiş bir kısıt dilinde bir ifade şeklinde yazılır.

UML üstmodelinde, sterotip üstvarlığı üstmodelin bir üstelemanı olan *GenişletilebilirEleman* ("*GeneralizableElements*") türünde olması sebebiyle Genelleştirme ("*generalization"*) ve Bağlılık ("*dependency*") ilişkilerine katılabilirler. PSM4WSS profili sterotipler arasında Genelleştirme ilişkisini sıkça kullanmaktadır. Sterotipler, UML'nin bir üstelemanı olan Sınıflayıcı ("*classifiers*") türünden olmadıklarından İlişkilendirme ("*association*") ilişkisine katılamazlar. Soyut bir *GenişletilebilirEleman* bir UML modelinde örneklenemez. Soyut sterotipler ilgili sterotipler arasında genel özellikleri genellemesi açısından kullanılabilirler. PSM4WSS soyut sterotipler kullanmaktadır.

İçerik, gezinim, kullanıcı ve iş akışı modeli için UML sınıf diyagramı ("*class diagram*") notasyonu ve sunum paketi için UML 2.0 ile gelen bileşim yapısı ("*composite structure*") diyagramı notasyonunun kullanılabilmesine imkan veren bir profil türetilmiştir.

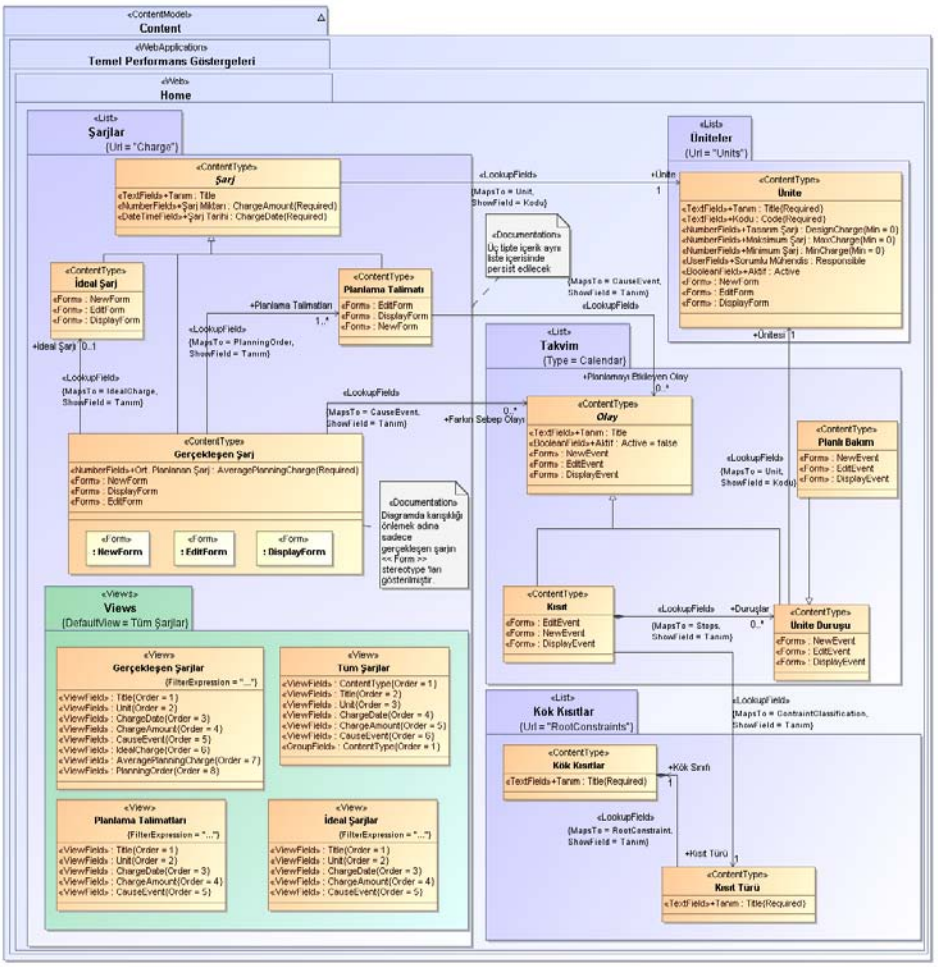
4 Örnek Senaryo

Senaryomuz, petrokimya endüstrisinde faaliyet gösteren bir kurumun üretim kaybı, kapasite kullanımı, üretim birimlerinin (üniteler) işletme emirlerini yerine getirebilme oranı gibi temel performans göstergelerinin (*"key performance indicators*") *(KPI*) belirlenmesinde planlama ve üretim kesimlerinden gerekli bilginin toplanması ve işlenmesini yöneten bir web sistemi çözümü gerçekleştirimidir. Planlanan, ideal ve gerçekleşen şarjların takibi yanında, gerçekleşen şarjların planlanandan sapmasının kurum içerisinde gerçekleşen olaylar ile –kök-sebep çözümlemesi ("*root cause analysis*") yapılarak sınıflandırılmış– ilişkilendirilmesi sistemin işlevsel ihtiyaçlarıdır.

Tüm alan kavramları, detaylı bir ihtiyaç mühendisliğinin ardından alan parçalaması ("*domain decomposition*") yöntemi ile şarjlar –ideal şarj, planlama talimatı ve gerçekleşen şarj–, olaylar –kısıt, ünite duruşu ve planlı bakım–, kök kısıtlar –kök kısıt ve kısıt sınıfı– ve üniteler –ünite– olarak farklı seviyelerde sınıflandırılmıştır. Üst kavramlar WSSNM'deki listelere, alt kavramlar ise içerik tiplerine eşleştirilmiştir. Mümkünse bir liste altına yerleşen içerik türlerindeki ortak özellikler, soyut birer süper sınıf (şarj, olay) içerisindeki alt-alanlar olarak toparlanmıştır. İçerik tipleri arasındaki ilişkilendirme ise WSSNM'deki arama alanının ("*lookup-field*") soyutlaması ile tamamlanır. Şekil 4'te söz konusu örnek senaryonun içerik yapısının PSM4WSS

üstmodeline uygun UML modeli verilmiştir. Bu içerik modeline dayalı olarak bir gezinim modeli ve sunum modeli ihtiyaçlara göre türetilir. Yer kısıtları nedeniyle bu bildiride diğer modelleri oluşturan tüm kavramların tanıtılması ve bu kavramlara dayalı olarak PSM4WSS görünümlerinin modellenmesi detaylandırılamamıştır.

İçerik modelinin XMI çıktısı, PSM4WSS modellerini işleyebilen bir uygulama sunucusuna diğer görünümlerin modelleri ile birlikte verildiğinde, modelden koda dönüşümler sonrasında tam otomatik olarak özel çözümün içeriği üretilir ve kullanıma hazır hale gelir. Şekil 2'de platforma özgü model seviyesi kısmında PSM4WSS modellerinden WSSNM'nin türetilmesi ve web uygulamasının üretilmesi mimari açıdan gösterilmiştir (Şekil üzerinde örnek modellerin ismi *KPI* olarak görülmektedir).



Şekil 4. PSM4WSS içerik paketi UML profili uygulanmış bir kullanıcı örnek modeli.

PSM4WSS'e dayalı yazılım çatısı WSS platformunu hem genişletir hem de platformun uygulama sunucusunda çalışan örneğini kullanır. PSM4WSS modellerinden WSSNM

için bir web uygulaması kurmayı ("*deployment*") gerektiren modellerin çıkarılması ve gerekli dönüşümlerin yapılması PSM4WSS çatısı tarafından gerçekleştirilir. Bu modeller hem metin bazlı hem de .NET nesneleri olarak oluşturulabilir. Sonrasında üretilen modelleri girdi olarak kullanan WSS, alt yapının kurulumunu gerçekleştirir. Son olarak PSM4WSS'e dayalı çatı PSM4WSS'in modellerine uygun olarak web uygulamasının statik yapısının üretilmesini tamamlar. Davranışsal yapısı ise SharePoint'in alt yapısından gelen olay yakalayıcıları ("*Event Receiver*") ve/veya SharePoint'in iş akış modelleri ile 2 türlü gerçekleştirilir. Olay yakalayıcıları, bir içerik birimde (liste, liste öğesi gibi) önceden tanımlanmış bazı olayları (öğe yaratma, öğe silinmesi gibi) senkron veya asenkron olarak yakalayan ve uygulamaya özel kullanıcı kod bloklarını çalıştıran olay-bazlı bir programlama modelidir. İş akışları ise model güdümlü olarak bir iş süreci ("*business process*") ile tanımlanan akışın üzerindeki bir takım aktivitelerin sistem kullanıcıları ya da web servisleri tarafından tamamlanmasını koordine edebilen bir yapıdır. İş akışları ayrıca model güdümlü olarak "*Business Process Execution Language*"'ın (BPEL) [8] platforma özgü örnekleri olan SharePoint durum makinesi iş akışı ("*Sharepoint State Machine Workflow*") ya da Sharepoint sırasal iş akışı ("*Sharepoint Sequential Workflow*") şeklinde gerçekleştirilebilinir. PSM4WSS'in iş akış modeli ("*WorkflowModel*") ile Sharepoint iş akışlarının entegrasyonu diğer görünümler ile (*NavigationModel, ContentModel, vb.*) sağlanır.

5 İlgili Çalışmalar

Literatürde WSS'in kendi nesne modeli ve onun XML şeması dışında web mühendisliği kavramlarını ve bakış açılarını içeren platforma özgü bir üstmodel bulunamamıştır. İlgili çalışmalar kapsamında daha çok platform bağımsız üstmodel önerilerinin olduğu söylenebilir.

Web mühendisliği alanında çalışan araştırmacıların geliştirdikleri alana özgü dillerini ve web modellerini tanımlamak için UML genişletme ve üstmodellemeden faydalandıkları görülmektedir (örneğin [9, 10, 11]). Conallen [9] standart UML diyagramları ile bağlantılı olarak, sunucu sayfaları ("*server pages*"), kullanıcı sayfaları ("*client pages*"), formlar, iskeletler ("*frames*") gibi tipik web ürünlerini ("*artifacts*") içeren Web uygulama genişletmelerini ("*Web Application Extensions*" - WAE) önermektedir. Ancak WAE'nin gezinim ve sunum bakış açılarının yüksek seviyeli soyutlamasının yeterli olmamasından hareketle Go´omez ve Cachero [1] web ara-yüz modeli sağlamak için UML diyagramlarını genişleten yeni bakış açısı önermiştir.

WebML [3] ve UWE [12] önerilerinde, web uygulamalarının tasarımı ve geliştirilmesinde içerik, gezinim ve sunum gibi ayrık modeller bazında farklı ilgilere işaret edilmektedir. Moreno ve arkadaşları [3] WebML ("*Web Modeling Language*") yaklaşımında, kendilerine ait alana özgü bir modelleme ortamı ve kod üretme dili tanımlamışlardır. UWE yaklaşımı [2, 12], UML uyumluluğu ve ticari platformların üstmodelleri içerisinde ortak kavramları bünyesinde bulundurması sebebiyle diğer web mühendisliği metotları içerisinde kendini ayırt etmektedir. Bu platform bağımsız üstmodeller arasında genel bir değerlendirme yapacak olursak UWE içlerinde MOF-uyumlu bir üstmodele ve UML profiline sahip olması, diğer üstmodellere kıyasla web mühendisliği alanında farklı çalışmalarda sunulmuş üstvarlıkların genelini içermesi ve sunduğu bakış açıları ve model elemanları ile öne çıkmaktadır.

6 Sonuç ve İleriye Yönelik Çalışmalar

PSM4WSS isimli platforma özgü bir web mühendisliği üstmodeli tanıtılmıştır. Microsoft SharePoint uygulama çatısı üzerinde geliştirilen ve MGM'ye dayanan bir yöntem ile önerilen üstmodelin kullanıcı modellerinden web tabanlı kurumsal çözümlerin otomatik olarak üretilmesi sağlanmaktadır. İleriye yönelik planlanan çalışma PSM4WSS ile UWE [12] üstmodeli arasında model dönüşümlerini sağlamaktır. Böylece web servislerine dayalı yazılım sistemlerinin eksiksiz bir model güdümlü yazılım geliştirme yöntemine uygun olarak önce platform bağımsız seviyede tasarlanması ve otomatik model dönüşümleri ve kod üretimleri sonrasında WSS çerçevesi üzerinde gerçekleştirimi mümkün olacaktır. PSM4WSS ile UWE üstvarlıkları arasındaki eşlemeler şu ana kadarki çalışmalarımız kapsamında tamamlanmıştır. Eşlemelere dayalı dönüşüm kurallarının yazılması hedeflenen ilk çalışmadır.

Kaynakça

1. Go´mez, J., ve Cachero, C., Information modeling for internet applications, Chapter OO-H method: extending UML to model web Interfaces, Idea Group Publishing, Hershey, PA, USA, 2003, s.144–173
2. Kraus, A., ve Koch, N., A metamodel for UWE, PhD thesis, Ludwig-Maximilians-Universitaet München, Ocak 2003.
3. Moreno N., Fraternalli, P., ve Vallecillo, A., WebML Modeling in UML, Institution of Engineering and Technology Software, Vol. 1, No. 3, Haziran 2007, s. 67-80.
4. Microsoft Sharepoint 2010, http://sharepoint.microsoft.com/, Microsoft (Erişim Tarihi: Ağustos 2010)
5. Model Driven Architecture Guide V.1.0.1, Object Management Group, OMG Document Number: omg/2003-06-01.
6. Meta Object Facility Specification, Object Management Group, OMG doc. AD/97-08-14, http://www.omg.org/docs/ad/97-08-14.pdf.
7. IBM Rational Software Architect (RSA) Standard Edition, http://www-01.ibm.com/software/awdtools/swarchitect/standard/features/ (Erişim tarihi: Nisan 2010)
8. Web Services Business Process Execution Language Version 2.0 (WSBPEL), OASIS Standard, April 2007, http://docs.oasis-open .org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf
9. Conallen, J., Building web applications with UML, Addison Wesley, 2002, 2nd edn.
10. Baresi, L., Garzotto, F., ve Maritati,M., W2000 as a MOF Metamodel. Proc. 6th WorldMulti-Conf. on Systemics, Cybernetics and Informatics - Web Engineering Track, Orlando, USA, 2002
11. Schauerhuber, A., Wimmer, M., ve Kapsammer, E., Bridging existing web modeling languages to model-driven engineering: a metamodel for WebML. Proc. of Model Driven Web Engineering, Palo Alto, CA, 2006.
12. Koch, N., Knapp, A., Zhang, G., ve Baumeister, H., UML-based Web Engineering: An Approach based on Standards (book chapter). In Web Engineering: Modelling and Implementing Web Applications. Springer, HCI, 2007

Gömülü Sistemler - III

Gömülü Yazılımlar için Model-Tabanlı Bir Bileşen Referans Modeli (UML-CM)

Mustafa DURSUN[1]

[1] ASELSAN A.Ş., Ankara
[1] mdursun@aselsan.com.tr

Özet. Bileşen tabanlı yazılım geliştirme bileşenlerin tekrar-kulanımını esas alarak güvenilir ve hızlı bir biçimde, maliyetlerini düşürerek yazılım geliştirmeyi adresler. Model-tabanlı yazılım geliştirme yaklaşımı, yüksek soyutlama seviyesinde Yazılım geliştirmeye imkan sağlamaktadırç. Model-tabanlı geliştirmenin bileşen tabanlı geliştirme ile beraber kullanılması bileşen tabanlı geliştirmenin maliyetleri düşürme hedefine tekra kullanımı üst seviye soyutlamada mümkün kılarak katkıda bulunur. Gömülü sistemlerin kısıtları gömülü yazılımların geliştirilmesinde mevcut bileşen referans modellerinin kullanımını zorlaştırmaktadır. Bu bildiride, gömülü sistemlerin kısıstları dahilinde bileşen-tabanlı yazılım geliştirmeyi model seviyesinde gerçeklemeyi hedefleyen gömülü sistemler için bir bileşen referans modeli ve modelin kontrol yazılımları ile kesişim alanı servislerinde kullanımı sunulmaktadır.

1 Giriş

Bileşen-Tabanlı Yazılım Geliştirme (BTYG), yazılım bileşenlerinin tekrar-kullanımı ile hızlı bir biçimde güvenilir yazılım geliştirmeyi hedeflemektedir[1]. Bileşen-Tabanlı Yazılım Geliştirme'nin temelinde tekrar-kullanılabilir yazılım bileşenleri vardır. Bir tekrar-kullanılabilir yazılım bileşeni en genel tanımıyla tam olarak geliştirilmiş ve bir yazılım ürünü için entegre edilebilir bir olgudur. Tekrar-kullanılabilir yazılım bileşenlerinin seçimi, uyarlanması ve entegrasyonu bir bileşen referans modeli(mimarisi) tarafından tanımlanmaktadır. Bir bileşen modeli; bileşenlerin ne olduğunu, semantik olarak nasıl yapılandırılmaları gerektiğini ve nasıl bileştirileceklerini tanımlar[8]. En çok bilinen bileşen modelleri EJB (Enterprise Java Beans)[3], COM [4] ve CCM (CORBA Component Model)[5] 'dir.

Bir bileşenin tekrar-kullanımından sağlanan yarar bileşenin soyutlama seviyesi ile orantılıdır. Soyutlama seviyesi yüksek bileşenler projenin daha erken safhalarında kullanılabilirler. Analiz ve tasarım en soyut tekrar-kullanım olgularıdır [6]. Model-Tabanlı geliştirme yazılım geliştiricilere yüksek soyutlama seviyelerinde çalışma imkanını sunar [7]. Tekrar-kullanılabilir bileşenlerin geliştirilmesinde ve kullanılmasında model-tabanlı geliştirmenin kullanılması bileşen-tabanlı geliştirmenin masraf düşürme özelliğine ciddi katkıda bulunur. UML [8], kullanımı gittikçe artan nesne-yönelimli geliştirmenin modellenmesinde kullanılan bir modelleme dili standartıdır [9,10]. UML araçları model-merkezli yöntemi kullanarak model-tabanlı

geliştirmeyi etkin bir biçimde yapabilmektedir[14]. UML, bileşen-tabanlı geliştirmeyi mümkün kılacak model elemanlarını da tanımlamaktadır.

Model tabanlı yazılım geliştirmeyi model-merkezli biçimde mümkün kılan UML araçlarının, gömülü gerçek-zamanlı yazılımların geliştirilmesinde etkin bir biçimde kullanılmasına rağmen EJB, COM ve CCM bileşen modelleri model tabanlı gömülü yazılımlarda etkin bir biçimde kullanılamamaktadır. Bunun sebebi bu modellerin çalışma ortamlarının ya gömülü sistemler tarafından desteklenmemesi ya da çalışma ortamlarının gömülü sistemlerin zaman veya bellek kısıtlarına uymamasıdır. Bileşenlerin, UML model elemanları kullanılarak geliştirilmesine çeşitli çalışmalarda yer verilmiştir [11, 12, 13]. Ancak bu çalışmalarda bileşen-tabanlı geliştirme bir bileşen referans modeli çerçevesinde tanımlanmamıştır.

Bu bildiride gömülü yazılımların bileşen tabanlı geliştirilmesi için model tabanlı bir bileşen referans modeli sunulmaktadır. Model, REHİS-GYM bünyesinde yapılan çalışma kapsamında [15] 2007 yılı sonunda ilk kez sunulmuş, bu süre zarfında gömülü kontrol yazılımları ve kesişim alanı servislerinde kullanılmıştır. Referans model, tekrar-kullanılabilir bileşenlerin tanımının, yapılandırılmasının ve bileştirilmelerinin UML model elemanları ile nasıl gerçekleneceğini tanımlamaktadır. UML Bileşen Referans Modeli, model-tabanlı yazılım geliştirmede, tekrar-kullanılabilir bileşenlerin kullanımını mümkün kılarak, yazılım geliştirme zamanının kısaltılması, test edilmiş olguların değiştirilmeden tekrar kullanımı, tekrar-kullanılabilir bileşenlerin kullanımlarından bağımsız olarak eş zamanlı geliştirilmelerini sağlar.

Bildirinin 2. ve 3. bölümlerde Bileşen-Tabanlı Geliştirme ve UML model elemanları, 4. bölümünde Bileşen Referans modeli, 5. bölümde Kontrol Yazılımlarında ve Kesişim Alanı Servislerinde Kullanımı, 6.bölümde ise sonuç aktarılmıştır.

2 Bileşen Tabanlı Yazılım Geliştirme

Bileşen-Tabanlı Yazılım Geliştirme (BTYG) tekrar-kullanılabilir yazılım bileşenlerinin seçimi, uyarlanması ve kurgulanmasını sağlayarak karmaşık yazılım sistemlerinin azami mühendislik gayreti ve kaynak harcaması ile hızlı bir şekilde geliştirilmesini amaçlayan bir yazılım geliştirme yaklaşımıdır [16]. Bileşenin tanımı konusunda bir fikir birliği söz konusu değildir. Bileşenin tanımı, kullanılan bileşen referans modeline ve bileşen teknolojisine göre değişmektedir[17]. Bileşen tanımının değişmesine rağmen, bileşenlerin ortak özelliklerinden söz etmek mümkündür. Bu özellikler, bağımsız geliştirilebilme, kara-kutu olarak kullanılabilme, sunulan ve gereken iyi-tanımlı arayüzlere sahip olma ve diğer bileşenler ile bileştirilebilmedir. Bileşenin tanımını, yapılandırılmasını ve bileştirilebilmesini ise bileşen referans modeli tanımlamaktadır. Bileşen referans modeli bileşenlerin geliştirilme sürecini adreslemezken bileşenler kullanılarak yeni yazılım sistemleri oluşturulmasını adresler.

Bileşen bileşimi, yeni bir sistem geliştirmek için bileşenlerin işlevsel olarak bir araya getirilmesi, kurgulanmasıdır [18]. Bileşen bileşimi bu özelliği ile bileşen referans modelinin sağlaması gereken en temel adımdır. Bileşen bileşimi, bileşenlerin ilişkilendirilmesi ile sağlanabilmektedir. İlişkilendirme bir bileşenin diğer bir bileşenin referansını elde etmesini sağlayan işlemdir [19]. Üç olası ilişkilendirme zamanından

bahsetmek mümkündür: Derleme-zamanı, bağlanma-zamanı ve yürütüm zamanı. Erken ilişkilendirme zamanları daha iyi statik analiz ve sistem optimizasyonuna izin verirken, geç ilişkilendirme zamanları sistem konfigurasyonunun kullanıcı tarafından yapılmasına ve konuşlandırma sonrası güncellemelere imkan verir [20]. Derleme-zamanı ilişkilendirme gömülü sistemler gibi kısıtlı kaynaklı sistemler için daha uygun iken, masaüstü ve sunucu sistemleri yürütüm zamanı ilişkilendirmeyi daha kolay destekleyebilmektedir [20].

Bileşen bileşimi, bileşenlerin sahip olduğu arayüzlerin birbirleri ile bağlanması ile gerçekleştirilir[21]. Bunun için bir bileşenin gereken arayüzünün diğer bileşenin sağlanan arayüzü ile bağlanması gerekir. Port, bir bileşenin arayüzlerini teşhir ettiği bir erişim noktasıdır [21] ve bileşenlerin birbirleri ile haberleşmelerini sağlar [22]. İki bileşen arasında etkileşimin sağlanması için bir bileşenin portundaki gereken arayüz ile bir diğer bileşenin portundaki bağdaşır sunulan arayüzün ilişkilendirilmesi gerekir [17] . Bağlaç iki bileşen arasındaki haberleşmedir [2]. Bu haberleşme iki bileşen arasında fonksiyon çağrımına izin verecek bir bağlantı olabileceği gibi karmaşık bir arakatman bileşeni de olabilir [2].

Bileşen referans modeli bileşenin ne olduğunu, bileşenlerin nasıl yapılandırılacaklarını ve bileştirileceklerini tanımlar. Bu çerçevede, bir bileşen referans modeli bir bileşenin tanımını, bileşen ilişkilendirme yöntemini ve zamanını, bileşen bağlacının tanımını ortaya koymak zorundadır. Değişik bileşen referans modelleri için bu tanımlar ve yöntemler amaçlanan alanda kullanılacak teknolojilere göre değişiklik göstermektedir.

3. UML Bileşen-Geliştirme Model Eleman Tanımları

UML, Bileşen-Tabanlı yazılım geliştirme için gerekli elemanların belirtimini yapmıştır[9]. Bu elemanların tanımları Bileşen Model Elemanları ve Bileşik Yapılar Model Elemanları başlıkları altında özetlenmiştir. Bileşenlerin modellenmesi UML model elemanlarının çeşitli şekilde kullanımı ile mümkündür [9]. Bileşen model elemanları başlığı altında bileşenin UML tanımı ve simgelemi, bileşik yapılar model elemanları başlığı altında model elemanlarının bileştirilmesi için gerekli elemanların tanımları ve simgelemleri verilmiştir.

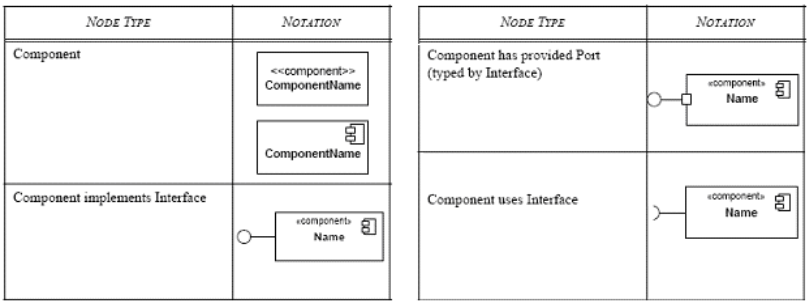
3.1 Bileşen Model Elemanları

UML bileşeni iyi-tanımlı arayüzlere sahip, belirli sayıda sınıflandırıcının durumunu ve davranışını kapsülleyen, modüler, özerk ve değiştirilebilir bir birim olarak tanımlar:

- Bileşen çevresi içinde değiştirilebilir olmalıdır.
- Bileşen belirli sayıda sınıflandırıcının durumunu ve davranışını kapsülleyebilecek özerkliğe sahip olmalıdır.
- Bileşen, tasarım veya yürütüm zamanında arayüzlerinin bağdaşırlığı ile aynı işlevselliği sunan başka bir bileşenin yerine kullanılabilir olmalıdır.

UML hem mantıksal (iş süreç bileşenleri) hem de fiziksel bileşenlerin (EJB, CORBA, .NET bileşenleri) belirtimini destekler. UML iki tip bileşenden bahseder: Temel Bileşenler ve Paketleme Bileşenleri. Temel bileşenler bir sistemde yürütülür bir eleman olarak tanımlanır. UML belirtiminde bileşenlerin ilişkilendirilmesi için özelleşmiş

bağlaçlar tanımlanır. Paketleme bileşenleri ise geliştirme sürecinin bir parçası olarak elemanların uyumlu grupları olarak tanımlanır. Paketleme bileşenleri, bileşenlerin yapı bloğu olarak tanımlanabilmesi için temel bileşen kavramının genişletilmesi sonucu doğmuştur. Bileşen model elemanlarının simgelemi Şekil 1'de verilmiştir.

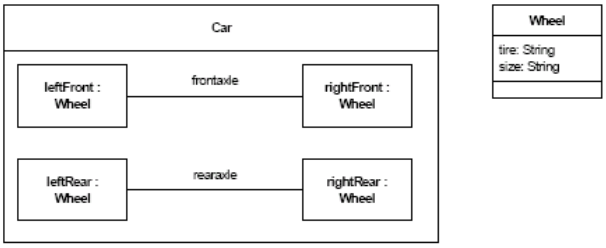


Şekil 1 Bileşen Model Elemanlarının Simgelemleri [9]

3.2 Bileşik Yapılar ve Arayüz Model Elemanları

Yapı terimi, birbirine bağlanmış elemanların bileşiminden bahseder. Birbirine bağlanmış elemanlar, yürütüm zamanı örneklerinin haberleşme bağlantıları üzerinden birlikte çalışmasını betimler. Bileşik Yapılar model elemanları BTYG kapsamaında gerekli olan yapılı sınıflandırıcı, port ve bağlaç model elemanlarını tanımlamaktadır.

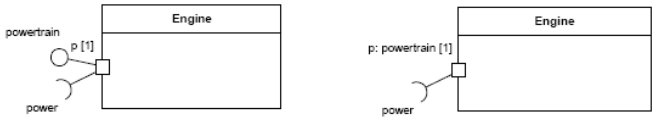
Bir yapılı sınıflandırıcı, davranışı tamamen veya kısımsal olarak sahip olduğu veya referansladığı birlikte çalışan örneklerce tanımlanan bir sınıflandırıcıdır. Yapılı sınıflandırıcının sahip olduğu örnekler kısım olarak tanımlanmıştır. Şekil 2'de gösterilen örnekte dört tekerleğe sahip araba bir yapılı sınıflandırıcıdır.Bu yapılı sınıf tekerlek örneklerinin birbirleri ile ilişkilendirilmelerine imkan sağlamakla birlikte bir araba örneği yaratıldığında dört tekerlek örneği ve tekerlekler arasındaki bağlantı da yaratılmaktadır.



Şekil 2 Bir Yapılı Sınıflandırıcı Örneği [9]

Bağlaç iki veya daha fazla örnek arasındaki haberleşmeye imkan veren bağlantıyı belirtir. Bu bağlantı bir işaretçi kadar basit bir gerçeklemeye sahip olabileceği gibi daha karmaşık bir ağ bağlantısı gerçeklemesi de olabilir.

Port bir sınıflandırıcı ile çevresi veya iç sınıflandırıcıları arasındaki ayrı erişim noktasını belirten bir sınıflandırıcı özelliğidir. Port bir sınıflandırıcının çevresine sunduğu ve çevresinden beklediği servisleri belirtir. Bir portun simgelemi Şekil 3'de verilmiştir.



Şekil 3 Port Kullanımı [9]

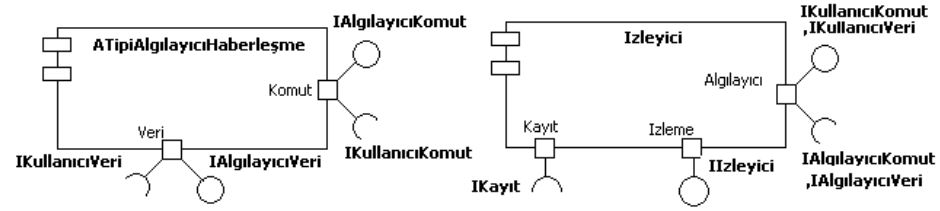
Bir portun davranış özelliği, portun belirttiği sunulan servisin portun sahibi sınıflandırıcı tarafından kotarıldığını belirler. Eğer bir port davranış portu ise sunulan servis porta sahip sınıflandırıcı tarafından kotarılıyordur.

Arayüz, diğer kullanıcılara açık öznitelik ve yükümlülükler bütününün bildirimini betimleyen bir sınıflandırıcı türüdür. Bir arayüzü bir kontratı belirtir, ve arayüzü gerçekleyen bir sınıflandırıcının herhangi bir örneği bu kontratı yerine getirir. Bir port kendisine sahip olan sınıflandırıcının çevresine sunduğu ve çevresinden beklediği servisleri arayüzler aracılığıyla betimler.

4. UML Bileşen Referans Modeli (UML-CM)

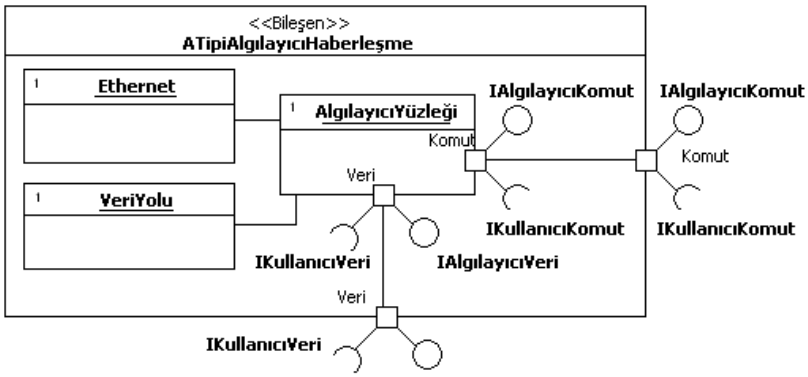
UML Bileşen referans modeli (UML-CM) gömülü yazılımlarda tekrar-kullanımı arttırabilmek için bileşenleri, bileşen yapılandırmasını ve bileştirmesini UML modelleri ile tanımlayan bir bileşen modelidir.

UML-CM, bileşeni tekrar-kullanılabilir yapı blokları olarak tanımlamaktadır. UML-CM bileşeni iyi tanımlı arayüzlere sahip, bağımsız ve tam geliştirilmiş, kara-kutu olarak kullanılabilir, test edilmiş ve belgelendirilmiş olmalıdır. UML-CM bileşeni bir sistem veya altsistemde bağımsız bir birimdir [10], sistem veya altsistemler bileşenlerden oluşabilir. UML-CM bileşen tanımı UML belirtiminde belirtilen tasarım zamanında kullanılan paketleme bileşenleri ile sınırlıdır. Bu bağlamda UML-CM bileşenleri kütüphane olgusunu üretebilen bileşenlerdir. Bileşen için kütüphane olgusunun üretilmesi ve model elemanları ile tekrar kullanılması bileşenin kullanıcı tarafından değiştirilmesini imkansız kılmakta ve bileşen için yapılmış testlerin geçerliliğini korumasını sağlamaktadır. Ayrıca bileşenlerin tekrar kullanımında derleme için harcanan zamanı ortadan kaldırmakta ve geliştirme zamanını hızlandırmaktadır. Şekil 4'de bir A tipi algılayıcı haberleşme bileşeni ve bir izleyici bileşeninin bileşen simgelemi gösterilmektedir. ATipiAlgılayıcıHaberleşme A tipi algılayıcı haberleşme bileşeni IAlgılayıcıKomut, IAlgılayıcıVeri arayüzünü uygularken IKullanıcıKomut, IKullanıcıVeri arayüzlerini kullanmaktadır. ATipiAlgılayıcıHaberleşme bileşeni A tipi algılayıcılarla komut arayüzünü ethernet üzerinden sağlarken veri alışverişini veri yolu üzerinden gerçekleştirmektedir.



Şekil 4 Izleyici ve ATipiAlgılayıcıHaberleşme Bileşen Simgelemi

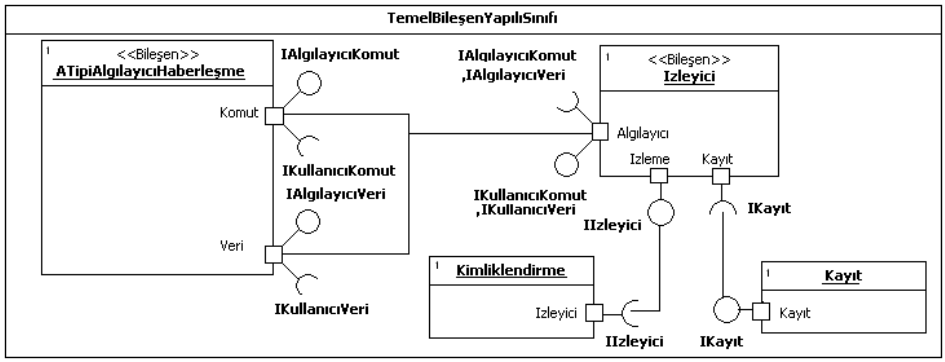
UML-CM bileşenleri yapısal sınıf kullanılarak yapılandırılır [13]. Yapısal sınıf bileşenin arayüzlerini, portlarını ve kısımlarını belgeler. Yapısal sınıf ayını zamanda bileşenin türediği sınıftır. Bileşenin davranışı oluşturacak sınıflar, yapısal bir sınıfın kısımları olarak tanımlanmaktadır. Bileşen yapısal sınıfının portları davranış portları değildir. Portlarda belirtilen servisleri yapısal sınıfın kısımlarını oluşturan sınıf örnekleri uygulamaktadır. Bu sınıfların sahip olduğu portlar davranışsal portlardır. Şekil 5'de ATipiAlgılayıcıHaberleşme bileşeninin yapısal sınıfı portları, arayüzleri ve kısımları verilmiştir.



Şekil 5 Bileşen Yapılandırması Yapılı Sınıf Diyagramı

UML-CM bileşenleri hiyerarşik bir yapıya sahiptir. Bir bileşen bileşeni başka bir bileşenin kısmı olabilir. Bu bileşenler dağıtık bileşenler değildir. Kısımları bağ ile ilişkilendirildiği için aynı düğüm üzerinde aynı bellek alanında çalışabilen kısımlara sahiptirler.

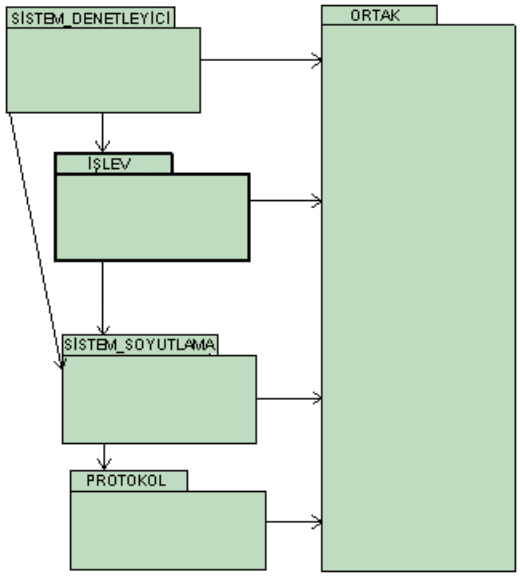
UML-CM bileşenleri yürütülebilir temel bileşeni oluşturmak amacıyla kullanılır. UML-CM bileşenlerinin bileştirilmesi, bileşenlerin yapısal sınıflarının örneklerinin temel bileşen yapısal sınıfında yaratılması ve bu örneklerin bağlar ile bağlantı-zamanında ilişkilendirilmesiyle gerçekleştirilir Temel bileşen yapısal sınıfı bileşenin tekrar-kullanım amacı olmadığından bileşen arayüzlerini ve portlarını belgelemez. Bileşenin yapılandırılmasında yapısal sınıfın kullanılması bileşenlerin yapısal sınıf örneklerinin, yapısal olmayan sınıf örnekleriyle de ilişkilendirilmesini mümkün kılmaktadır. Şekil 6'da ATipiAlgılayıcıHaberleşme ve Izleme bileşenlerinin bileştirilmesi örneği sunulmaktadır.



Şekil 6 Bileşen Bileştirilmesi Yapılı Sınıf Diyagramı

5. Bileşenlerin Kontrol Yazılımlarında ve Kesişim Alanı Servislerinde Kullanımı

ASELSAN Elektronik Harp ve Silah Sistemleri projelerinde kullanılmak üzere tasarlanan gömülü kontrol yazılımları mimarisinde [23], katmanlı mimari tasarım kalıbı kullanılmıştır. UML Bileşen Referans Modeli bileşenleri bu mimaride istenilen katmanda kullanılabilir özelliğe sahiptir. Ayrıca bileşenlerin hiyerarşi özelliği, bir alt katmandaki bileşeni üst katman içinde kullanmaya imkan vererek kullanıcıyı altkatman uygulama detaylarından soyutlamaktadır. UML Bileşen Referans Modelinin bu özelliği, protokol katmanı bilgisi olmadan yalnızca sistem soyutlama arayüzü bilgisine sahip olunarak kontrol yazılımı geliştirmeyi mümkün kılmaktadır.



Şekil 7 Katmanlar Arası İlişkiler [23]

Katmanlı mimari tasarımda alt seviyede yer alan Protokol ve Sistem Soyutlama katmanının az sayıda bağımlılığı olduğu görülürken üst katmanlarda bağımlılık sayısı artmaktadır. Bu durum alt seviyedeki bileşenlerin yeniden kullanılırlığı yüksekken, İşlev ve Sistem Denetleyici katmanındaki bileşenlerin yeniden kullanılırlık imkanının daha az olduğunun bir göstergesidir [23]. UML Bileşen Referans Modeli protokol ve soyutlama katmanının tekrar kullanımında etkin olarak kullanılabilmektedir. UML-CM bu katmalar için ayrı ayrı bileşenlerin geliştirilmesine olanak sağladığı gibi iki katman elemanlarını kapsayacak bir bileşen geliştirmeye de olanak sağlamaktadır.

UML Bileşen Referans Modeli bileşenleri Kesişim Alanı Servislerinden Hata Kaynak Yönetimi bileşenlerinin geliştirilmesinde aktif olarak kullanılmıştır[24]. Hata Kaynak Yönetimi kütüphanesi tamamiyle tekrar-kullanılabilir bileşenlerinden oluşmaktadır. Bu bileşenler Kontrol yazılımı mimarisi içinde kullanılabildiği gibi bu mimariden bağımsız, bileşen-yönelimli mimarilerde de kullanılabilmektedir.

6. Sonuç

Bileşen-tabanlı yazılım geliştirme (BTYG) bileşenlerin tekrar-kulanımını esas alarak güvenilir ve hızlı bir biçimde maliyetlerini düşürerek yazılım geliştirmeyi adresler. BTYG'nin temelinde tekrar-kullanılabilir yazılım bileşenleri vardır. Yazılım bileşenlerinin tekrar-kullanılabilirliği model seviyesinde bu bileşenlerin ele alınarak model-merkezli bir gelştirme yürüterek arttırılabilmektedir. Bu çalışmada gerçek-zamanlı ve gömülü sistemler için model seviyesinde yazılım bileşenini tanmlayan ve bu bileşenler ile BTYG'yi çözümleyen bir bileşen referans modeli, UML-CM, sunulmuştur.

Sunulan bileşen referans modeli ile sınıfların ve arayüzlerin kapsüllenmesi bileşen seviyesinde daha etkin bir biçimde gerçeklenebilmektedir. Nesne-yönelimli yazılım geliştirmede soyutlama seviyesi nesnelerde kalmakta ve nesnelerin kullanılması konuşlandırmanın modellenmesini sınırlandırmaktadır. Bir sınıfın tekrar kullanımı gerektiğinde o sınıfının bağımlı olduğu diğer elemanların da tekrar kullanıma dâhil edilmesi gerekmekte ve bu durum tekrar kullanılabilirliği azaltan bir etken olmaktadır.

Kullanıcıya bileşenler kara-kutu olarak sunulmakta ve kullanıcıya yanlızca bileşenin sisteme adaptasyonu ve bileşenin yapılandırılması işi kalmaktadır. Bileşenlerin kütüphane olgusunu üretebilir olması tekrar-kullanılabilirliği test, dokümantasyon ve derleme zamanlarını da içine alacak şekilde bileşen-tabanlı yazılım geliştirme esaslarına göre genişletmektedir. Bileşenlerin kara-kutu olarak kullanılması noktasında, bileşenlerin tekrar-kullanılabilirlini arttırmak için bileşenlerin ürün hattı yaklaşımına uygun bir biçimde geliştirilmesi gerekmektedir.

Bileşen referans modelinin tanımladığı yöntem ile kontrol yazılımları ve kesişim alanı servislerinde bileşen-tabanlı yazılım geliştirme gerçekleştirilmiştir.

Kaynakça

1. N.S. Gill, “Reusability Issues in Component-Based Development”, ACM SIGSOFT Software Engineering Notes,2003
2. R. Senthil, D. S. Kushwaha, A. K. Misra, “An Improved Component Model for Component based Software Engineering”, July 2007, ACM SIGSOFT Software Engineering Notes
3. http://java.sun.com/products/ejb/
4. www.microsoft.com/COM/
5. http://www.omg.org/technology/documents/formal/components.htm
6. Capt James E. Cardow, “Issues On Software Reuse”, 1989
7. http://www-07.ibm.com/hk/e-business/events/archives/2005/downloads/Practical_MDD_Seminar.pdf
8. http://www.uml.org/
9. Object Management Group,. Unified Modeling Language, Superstructure, v2.2, 2007-11-02 http://www.omg.org/cgi-bin/doc?formal/09-02-02
10. Object Management Group,. Unified Modeling Language, Infrastructure, v2.2, 2007-11-04 http://www.omg.org/cgi-bin/doc?formal/09-02-04
11. Bruce Powel Douglass, “UML 2.0: Incremental Improvements for Scalability and Architecture”, Telelogic White Paper, RTC Magazine 2003
12. Bruce Powel Douglass, “Real Time UML: Advances in the UML for Real-Time Systems”, Addison Wesley, 3rd Edition 2004
13. James Ivers, Paul C. Clements, David Garlan, Robert Nord, Bradley Schmerl, Jaime Oviedo-Silva, “Documenting Component and Connector Views with UML 2.0”, Technical Report CMU/SEI-2004-TR-008, 2004
14. Elmqvist, Simin Nadjm-Tehrani, “Model-Driven Software Development”, p 291-303, Jonas 2005
15. Mustafa DURSUN, “Rhapsody Gerçek-Zaman Çatısında Bileşen Tabanlı Geliştirme”, Proje Araştırma Raporu, 2007
16. O.P. Rotaru, M. Dobre, “Reusability Metrics for Software Components”, 2005
17. He Jifeng, Xiaoshan Li, Zhiming Liu, ”Component-Based Software Engineering “,ICTAC 2005, LNCS 3722, pp. 70–95, 2005.
18. Ian Sommerville, “Software Engineering”,Addison-Wesley, 2004
19. Alan Dearle, ”Software Deployment, Past, Present and Future” Future of Software Engineering(FOSE'07), 2007
20. Venkat Chakravarthy, John Regehr, Eric Eide, “Edicts: Implementing Features with Flexible Binding Times”, AOSD’08, March 31 – April 4, 2008
21. Desmond Farncis D’souza, Alan Cameron Wills, “Objects, Components, and Frameworks with UML: The Catalysis Approach”, Addison Wesley, 1999
22. T. Genßler, O. Nierstrasz, M. Winter, “Components for Embedded Software”, CASES 2002, October 8–11, 2002, Grenoble, France

23. H. Özgür Gören, E. Gürler, “Elektronik Harp Sistemleri Gömülü Kontrol Yazılım Mimarisi”, UYMK 2006

24. Metin Tekkalmaz, E.Gürler, Mustafa Dursun, “Görev Kritik ve Gömülü Sistemler icin T5D Uyumlu Bir Hata Yönetimi Altyapısı Tasarımı ve Gerçeklemesi”, UYMS 2009

eMON: Gerçek Zamanlı Gömülü Sistemlerin Çalışma Zamanı Görselleştirilmesi İçin Monitör Yazılımı

Berkant AKIN[1] Mehmet GÖKÇAY[2] Kaan DOĞAN[3]

[1,2,3] TÜBİTAK-SAGE Yazılım Birimi, Pk 16, Mamak 06261 Ankara / Türkiye
[1]berkantakin@yahoo.com, [2]mgokcay@gmail.com, [3]kaandogan@gmail.com

Özet. Gerçek Zamanlı gömülü sistemlerde hedef bilgisayarı/işlemciyi durdurmadan ve gerçek zamanda çalışmayı bozmadan gözlemlemek, yazılım davranışını anlamada ve hata ayıklama süresinin kısaltılmasında önemli bir yer tutmaktadır. Gömülü sistemi monitör edebilmek için gözlemlenecek sembollerin hedef bilgisayarda bulunduğu adresin, sembol tipinin ve hedef bilgisayar hafıza organizasyonunun bilinmesine ihtiyaç vardır. Bu çalışmada, sembol yönetimi dinamik ve statik olmak üzere iki şekilde yapılmaktadır. Dinamik sembol yönetimde sembol bilgileri, ELF/DWARF tipindeki hata ayıklama bilgisi çözülerek elde edilir. Bu yönetimde semboller çalışma zamanında hedef yazılımdan istenilen hızda örneklenip eMON-PC yazılımına aktarılarak görselleştirilmesi gerçekleştirilir. Statik sembol yönteminde ise yazılım içindeki veri paketleri XML tabanlı monitör tanımlama dosyaları ile tanımlanarak hedef yazılımdan istenilen hızda ve istenilen kod satırından eMON-PC yazılımına gönderilerek işlenir. eMON-PC yazılımı içerdiği görsel bileşenler ile hedef bilgisayardan toplanan verilerin etkin bir şekilde görselleştirmesi/yönlendirilmesi; XML tabanlı betik(Ing. Script) yazma sayesinde dinamik bileşen yaratma; Python betikleme sayesinde ise kullanıcıya toplanan veriler üzerinde işlem yapabilme olanağı sunmaktadır.

2 Giriş

eMON yazılımı gerçek zamanlı gömülü sistemin çalışma zamanında etkin bir şekilde gözleme ihtiyacından doğmuştur. Yazılımın gözlemlenebilmesi için gözlenilen parametrenin adres, tip ve boyut bilgilerine ihtiyaç vardır. Semboller basit tipler olabileceği gibi iç içe geçmiş karmaşık veri yapıları şeklinde de olabilir. eMON mimarisinde sembol yönetimi dinamik ve statik olarak gerçekleştirilmektedir.

Dinamik sembol çözümlemesinde eMON yazılımı ELF(Executable and Linkable Format)/DWARF(Debug With Arbitrary Record Format)[1] tipindeki çalıştırılabilir kod içerisinde bulunan hata ayıklama bilgisini kullanarak yazılıma bağlanmış sembol bilgileri elde edilir. Bu yöntem kullanıcıya herhangi bir anda hedef yazılım değiştirmeden ve durdurmadan istenilen parametreyi gözlemleme olanağı sunmaktadır. Burada önemli bir nokta nesne kodu içerisindeki hata ayıklama bilgisinin biçimi ve nasıl işleneceğidir. Yaygın olarak kullanılan hata ayıklama bilgi biçimleri STABS[2], DWARF[3],[4] ve .PDB(program database)[5] olarak sıralanabilir. IDE'ler bu bilgileri kullanarak hata ayıklama işlevlerini(watch, trace, breakpoint gibi) yerine getirirler. Esneklik, ölçeklendirilebilirlik ve dökümantasyon göz önüne alındığında DWARF hata ayıklama dosya biçiminin öne çıktığı görülmektedir. Texas Instruments gibi önemli işlemci üreticileri COFF(common object file format)/STABS formatının yanı sıra ELF/DWARF formatını da desteklemeye başlamıştır [6]. eMON yazılımı DWARF2/3/4 tipindeki hata ayıklama bilgisini işleyebilmektedir. Dinamik sembol yönetimi kullanan monitör yazılımına örnek olarak µProbe [7] yazılımı gösterilebilir. Micrium firması tarafından geliştirilen bu yazılım ELF ve IEEE 695 formatındaki nesne kodlarını işleyebilmektedir. eMON yazılımının betik tabanlı olması µProbe yazılımından farklarından biridir. Dinamik semboller kullanılarak veri örneklemenin önemli bir dezavantajı ise örnekleme anı dışında kalan zamanlarda oluşabilecek değişimlerin kaçırılabilme ihtimalidir. Bunu önlemek için örnekleme hızı üzerinde değişiklikler yapılabilir.

Statik sembol çözümlemesi ise herhangi bir hata ayıklama ve nesne kodu kullanmadan yazılım içerisindeki parametrelerin temel veri tipleri kullanılarak tanımlanması ilkesi üzerine kuruludur. Tanımlama XML ile yapılır ve her bir sembole benzersiz ID atanır. Sembol bilgileri hedef yazılım üzerinde çalışan eMON-Ghost modülü kullanılarak eMON-PC yazılımına aktarılır. eMON-PC yazılım sembol ID bilgisini XML dosyasındaki ilgili sembol ID'si ile eşleştirerek veri çözümlemesini gerçekleştirir. XML tabanlı veri tanımlama yönetimi sıkça

kullanılmaktadır. Telemetre sistemleri için geliştirilen XIDML[8] ve µProbe çip tanımlama dosyaları[7] buna örnek olarak verilebilir. eMON yazılımın kullandığı monitör tanımlama dosyası DWARF parametre tanımlama biçimiyle(ofset, taban tip boyutu gibi) uyumludur. Böylelikle dinamik ve statik sembol yönetimi aynı yazılım modülü tarafından işlenebilmektedir.

eMON yazılımın önemli özelliklerinden biri ise betik tabanlı olmasıdır. Görsel bileşenler ve semboller birbirlerine XML tabanlı betikler ile bağlanmaktadır. Bu bağlamda görsel arayüz programlanabilmektedir. Python ile betikler yazılarak hedef bilgisayardan toplanan veriler üzerinde işlem yapılabilir. Python ve eMON-PC Python C/C++ API [9] üzerinden birbirleri ile haberleşirler.

eMON-PC yazılımı gömülü sistemler için temel, aviyonik sistemler için ise özel görsel bileşenler içermektedir. Bu bileşenler:

Görsel: Grafik, Izgara, DirectX tabanlı 3D model gösterimi, 2D Bitmap ve DTED tipinde yükselti haritası, histogram, atış zarfı (LAR), HUD;

Haberleşme: Dosya kayıt, Dosya yükleme, UDP, SpaceWire, UART,

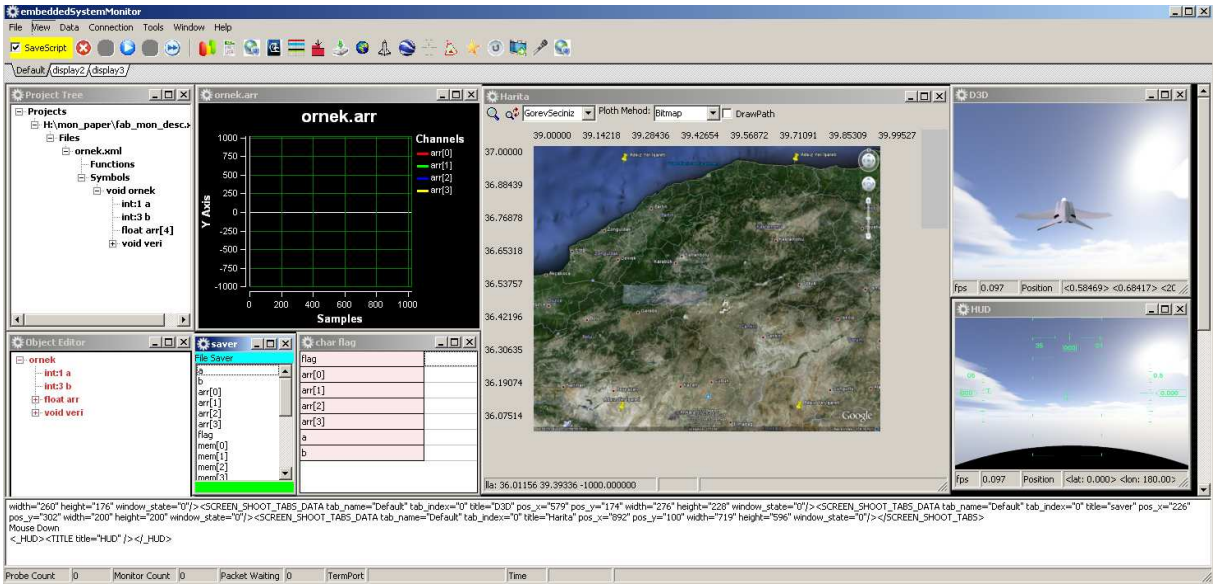
Middleware ve API'ler: Microsoft Speech Api (SAPI), GoogleEarth arayüzü.

3 eMON Mimarisi

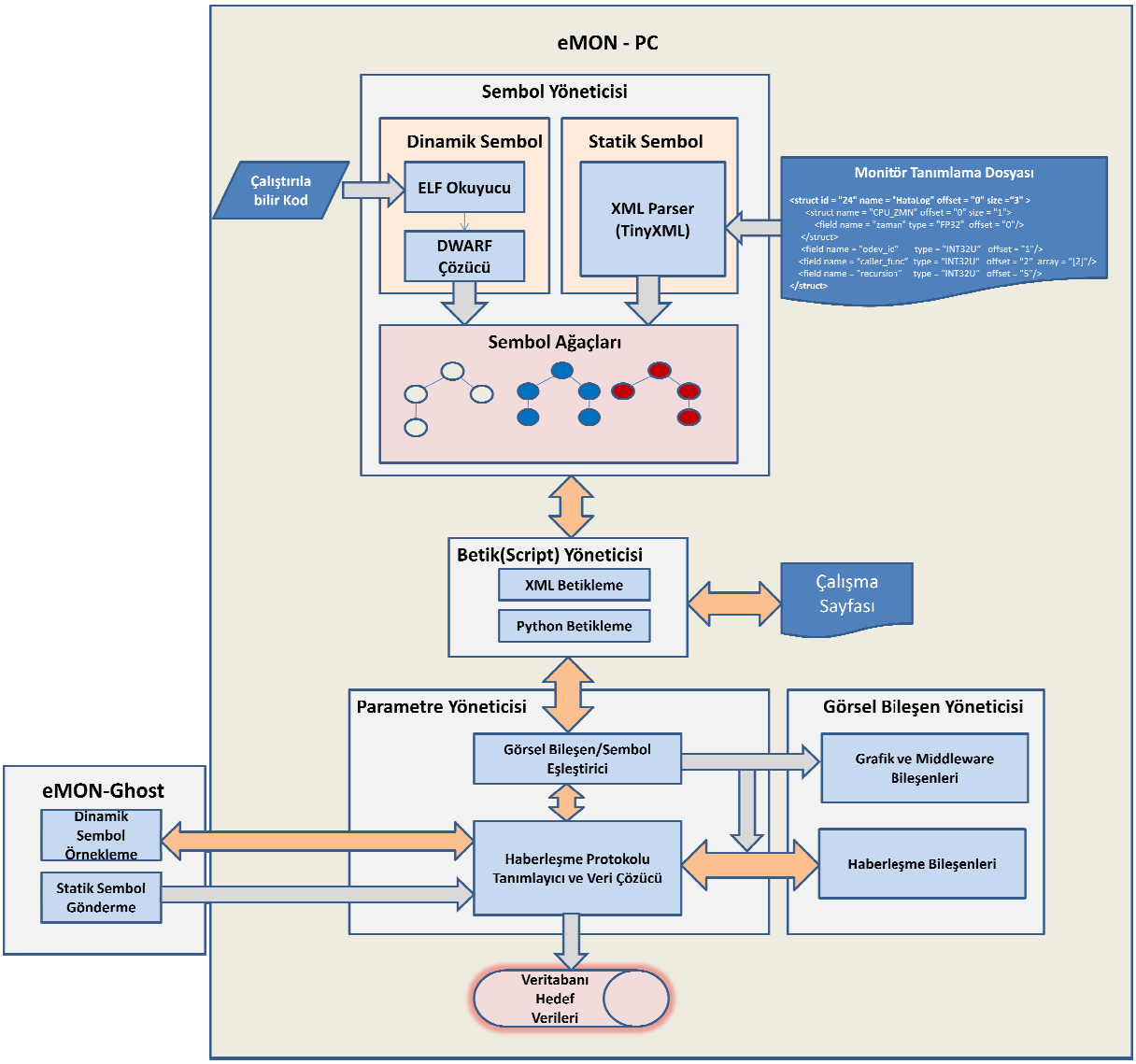
eMON yazılımı Microsoft Windows XP üzerinde çalışan C++ temelli eMON-PC ve hedef bilgisayar üzerinden çalışan C/C++ temelli eMON-Ghost yazılımlarından oluşmaktadır. eMON-PC ve eMON-Ghost herhangi bir veri yolu üzerinden birbirleri ile haberleşirler.

3.1 eMON-PC Mimarisi

eMON-PC yazılımında dinamik / statik sembol yönetimi, hedef bilgisayar ile haberleşme, sembollerin görsel bileşenlere bağlanması, XML / Python tabalı betikleme, çalışma alanı yönetimi(workspace) işlevlerini gerçekleştirmektedir. Sembollerinin görsel bileşene bağlanması sürükle-bırak mantığına göre yapılmaktadır. eMON-PC kullanıcı arayüzü ve mimarisi sırasıyla Şekil 1 ve Şekil 2'de verilmiştir.



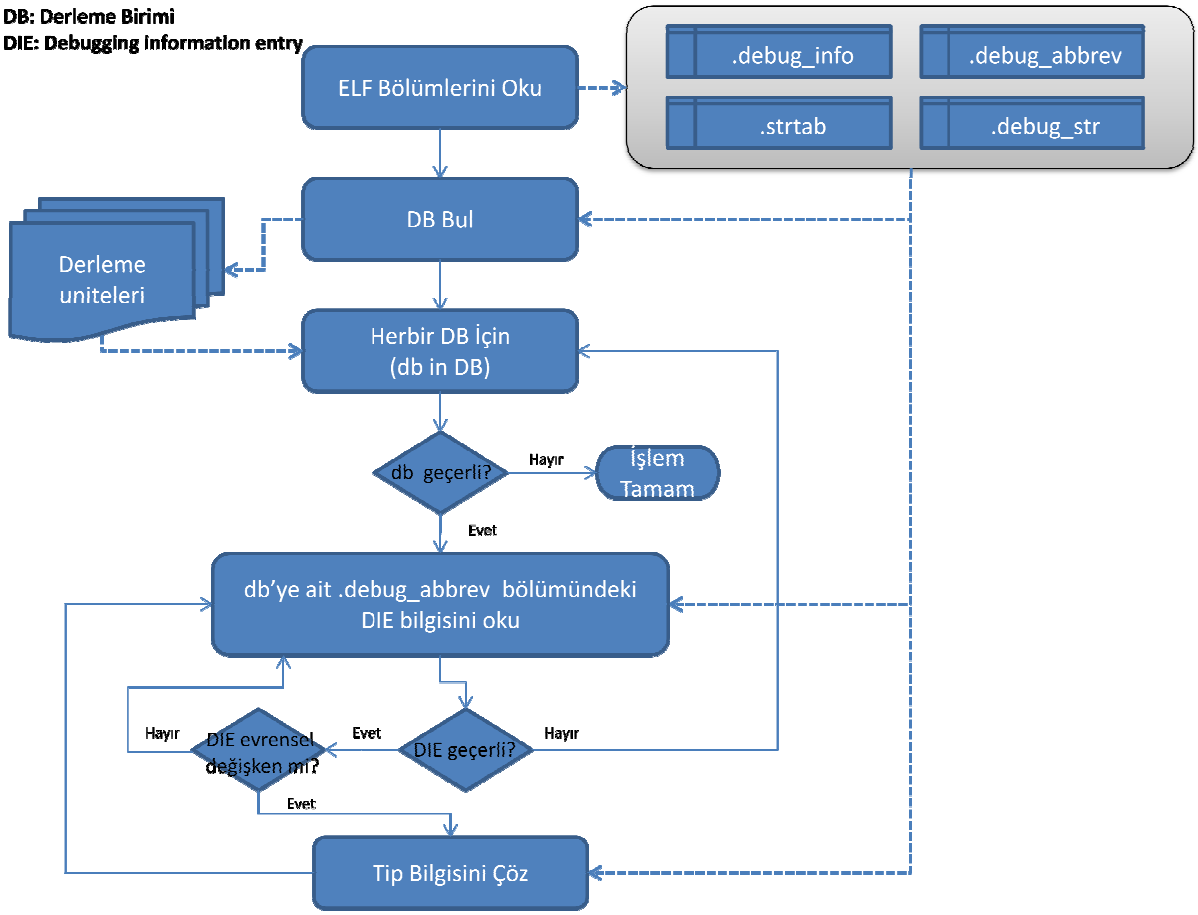
Şekil 1 eMON-PC GUI



Şekil 2 eMON Yazılım Mimarisi

3.2.1 Sembol Yöneticisi

Dinamik sembol yönetimi nesne kodu içerisindeki evrensel sembollerin adres ve tip bilgilerini çözerek sembol ağacına ekler. Sembol ağacına eklenen her bir elemanın adres boyut, tip, kodlama (int, float, double, char vb.) ve ata'ya göre ofset bilgileri DWARF formatındaki hata ayıklama bilgisinden çıkartılır. Dinamik semboller çalışma zamanında hedef bilgisayardan örneklendiği için hedef bilgisayarın hafıza organizasyonun bilinmesi gerekmektedir. DWARF bilgisi içerisinde tip bilgileri ve tip boyutları net bir şekilde tanımlandığı için hedef bilgisayarın hafıza adresleme mimarisi 8/16/32/64 bit olması herhangi bir şeyi değiştirmemektedir. ELF/DWADF biçimdeki nesne kodundan dinamik sembol çözümleme akış diyagramı Şekil 3'de verilmiştir.



Şekil 3 DWARF İçerisinde Dinamik Sembol Çözümleme

Statik sembol yönetimi ise XML biçimindeki monitör tanımlama dosyaları aracılığı ile yapılmaktadır. Bu dosya hedef bilgisayara ait mimarisel bilgiler ve veri tanımla bölümü olmak üzere iki ana bölümden oluşmaktadır.

Mimarisel bilgiler bölümünde hedef platformun bayt boyutu, temel tip, bu tiplerin boyutları ve tiplerin nasıl çözüleceği tanımlanmaktadır. Bayt boyutu genelde 8 bit olsa da çoğu DSP işlemcilerinde 32 bittir. Bayt boyutu tanımlanmasının ardından hedef platformdaki temel veri tiplerinin bayt büyüklüğü ve DWARF uyumlu veri kodlama tipinin tanımlanması gerekmektedir. DWARF uyumlu baze kodlama biçimleri Tablo 1'de[3] verilmiştir.

Tablo 1 DWARF uyumlu sembol kodlamaları

Name	Meaning
DW_ATE_address	linear machine address (for segmented addresses see Section
DW_ATE_boolean	true or false
DW_ATE_complex_float	complex binary floating-point number
DW_ATE_float	binary floating-point number
DW_ATE_signed	signed binary integer
DW_ATE_signed_char	signed character
DW_ATE_unsigned	unsigned binary integer
DW_ATE_decimal_float	decimal floating-point number

Sembol tanımlamaları ata-çocuk(Ing. parent-child) ilişkisi korunarak DWARF uyumlu olarak C/C++ dilinde yer alan struct, union, array ve bit alanı tanımlamalarına izin vermektedir. Aşağıda monitör tanımlama dosyası söz dizimi verilmiştir.

<**file** ***name*** = "dosya_ismi" >
<**architecture**>
<**byte_size**>[8|32]</byte_size>
<**types**>
<**type** ***name*** = "?" bytes = "1|2|4|8" encoding = "Tablo 1'deki herhangi bir tip"/>
.....
</**types**>
</**architecture**>
<**struct**|**union** ***id*** = "id" name = "sembol_ismi " ***offset*** = "0" ***size*** = "sembol_boyutu">
<**field** name = "isim1" ***type*** = "INT32U" [***bit_offset*** = "?"] [***bit_size*** = "?"] [***array*** = "[?][?]" ***offset*** = "?"/>
<**struct**|**union** ***name*** = "alt_sembol_ismi " ***offset*** = "0" ***size*** = "sembol_boyutu">
.....
</**struct**|**union**>
.....
</ **struct**|**union** >
……..
</**file**>

Örnek: Aşağıdaki veri yapısını 8 bitlik bayt boyutuna sahip platformda tanımlamak için gerekli monitör tanımlama dosyası aşağıda verilmiştir:

```
struct ornek {
        int a:1;
        int b:3;
        float arr[4];
        struct{
                char flag;
                double mem[16];
        }veri;
}
```

```
<file name = "ornek_mon_tanımlama">
  <architecture>
    <byte_size>8</byte_size>
    <types>
      <type name = "char"  bytes = "1" encoding =   "DW_ATE_signed_char"/>
      <type name = "int"   bytes = "4" encoding =   "DW_ATE_signed"/>
      <type name = "float" bytes = "4" encoding =   "DW_ATE_float"/>
      <type name = "double"bytes = "8" encoding =   "DW_ATE_double"/>
    </types>
  </architecture>
  <struct id = "0" name = "ornek "   offset = "0" size = "66">
    <field name = "a" type="int" bit_offset="0"bit_size ="1" offset = "0"/>
    <field name = "b" type="int" bit_offset ="1"bit_size ="3"offset = "0"/>
    <field name = "arr"   type = "float" array = "[4]" offset = "1"/>
    <struct  name = "veri " offset = "17" size = "49">
      <field name = "flag"   type = "char" offset = "0"/>
      <field name = "mem"    type = "double"  array = "[16]" offset = "1"/>
    </struct>
  </struct>
</file>
```

3.2.2 Betik Yöneticisi

Betik yöneticisi, görsel bileşenlerle sembolleri birbirlerine bağlamak ve bileşen verileri üzerinde işlem yapabilmek için geliştirilmiş otomasyon altyapısıdır.

XML Betik altyapısı, eMON-PC üzerindeki görsel bileşenlere sembol atamak için kullanılır. eMON-PC kullanıcı arayüzü kullanılarak yapılan herbir sembol-görsel bileşen ataması XML betik kodu oluşturularak gerçekleştirilir. Bu yapı sayesinde kullanıcı arayüzü kullanılmadan da istenilen betiklerin işletilmesi de mümkün olmaktadır.

Aşağıda "ornek.xml" dosyası içerisinde yer alan "ornek" yapısındaki "flag" parametresinin ızgara görsel bileşenine bağlayan XML betik kod parçası görülmektedir.

```
1.  <LOAD_MON_PROJECT project = "C:\ornek.xml" />
2.  <ADD_OBJECT project = "C:\ornek.xml" file = "ornek_mon_tanımlama" path = "ornek →veri→flag"/>
3.  <GRID>
4.    <TITLE title="FlagIzgara" grid_style="0"/>
5.          <GROUP_GRIDER project="C:\ornek.xml" file ="ornek_mon_tanımlama" path ="flag" grid_style= "0"/>
6.  </GRID>
```

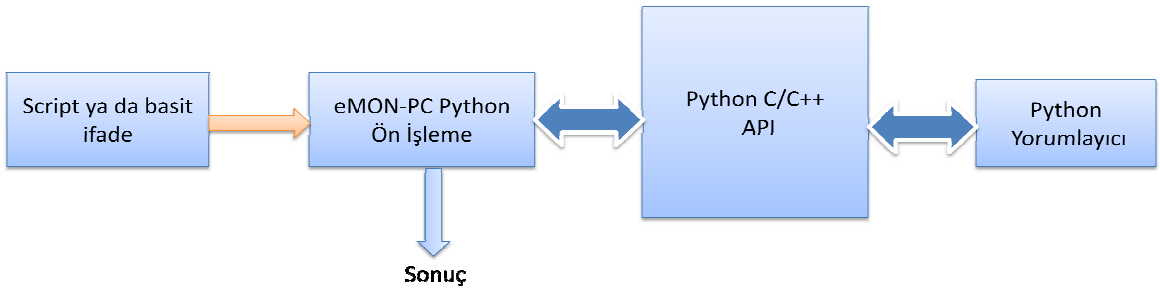
Satır 1'de monitör tanımlama dosyası yüklenir ve sembol ağacı oluşturulur. Satır 2'de "flag" parametresinin sembol ağacındaki yolu tanımlanarak nesne editörüne eklenir; Satır 3-6 arasında ise ızgara görsel bileşeni ile flag parametresinin bağlanması gerçekleştirilir.

XML betik arayüzünde yer alan görsel bileşen yaratma takıları ve işlevleri Tablo 2'de verilmiştir. Her bir takıya ait öznitelik (Ing. Attribute) seti yer sınırlamasından dolayı verilmemiştir.

Tablo 2 Görsel bileşen XML takıları

TAG	**İşlev**
PLOTTER	2D grafik
GRID	Izgara
STR_HIST	Histogram
FILE	Dosya kaydedicisi
FILE_LOAD	Dosya yükleyicisi
D3D	3D model
GE	Google Earth API kontrolü
HUD	HUD görütüsü oluşturur
LAR	Atış zarfı görüntüsü
UDP	UDP istemcisi yaratır
SPWR	SpaceWire istemcisi yaratır
SAPI	Konuşma API kontrolü yaratır
STATEDIAG	Durum geçiş diyagramı oluşturur.

Python Betik altyapısı ise görsel bileşenlerin aktivasyonu (trigger) ve görsel bileşenlerdeki veri üzerinde ekstra işlem yapmak için geliştirilmiştir. Python kullanabilmek için eMON-PC içerisinde Python yorumlayıcıya veri/ifade ve/veya kod geçirilmesi gerekmektedir. Bu bağlamda Python C/C++ API [9] kullanılarak eMON-PC ile Python yorumlayıcı arasında iletişim sağlanmıştır. eMON-PC ile Python yorumlayıcı arasındaki ilişki Şekil 4'de verilmiştir.



Şekil 4 eMON-PC Python Yorumlayıcı veri akışı

Görsel bileşen verisi kullanarak herhangi bir ifadeyi Python yorumlayıcısına gönderirken port numarasını ifade etmek için $ işareti kullanılır. Aşağıdaki aktivasyon betik örneğinde, port 1 ve port 2'deki verilerin toplamı 100'ü geçtiğinde görsel bileşenin aktifleştirilmesini sağlayan kod verilmiştir:

($0 + $1) > 100

Burada ”($0 + $1) > 100” ifadesi eMON-PC tarafından ön-işlemden geçirilir. $x yerine x numaralı portun verisi konularak Python yorumlayıcıya geçirilecek ifade elde edilir. *PyRun_SimpleString* gibi Python API fonksiyonları kullanılarak ifadenin sonucu hesaplanır. Sonuç yine Python C/C++ API kullanılarak yorumlayıcıdan eMON-PC adres uzayına aktarılır.

Herhangi bir bileşendeki verinin ilgili portundaki veriyi Python yorumlayıcısına aktarmak için aşağıdaki söz dizimi kullanılır:

var = #[bileşen ismi].$x

ornek.xml kod listesinde verilen “flag” parametresinin değerini Python yorumlayıcısına göndermek için aşağıdaki ifade işletilir:

flag_py = #FlagIzgara. $0

3.3.3 Parametre Yöneticisi

Parametre yöneticisi işlevleri aşağıda sıralanmıştır:

i. Sembol-görsel bileşen eşleştirilmesinin yapılması;
ii. Dinamik sembollerin gruplanması
iii. Hedef bilgisayardan alınan verinin DWARF kodlama bilgisine göre çözülmesi

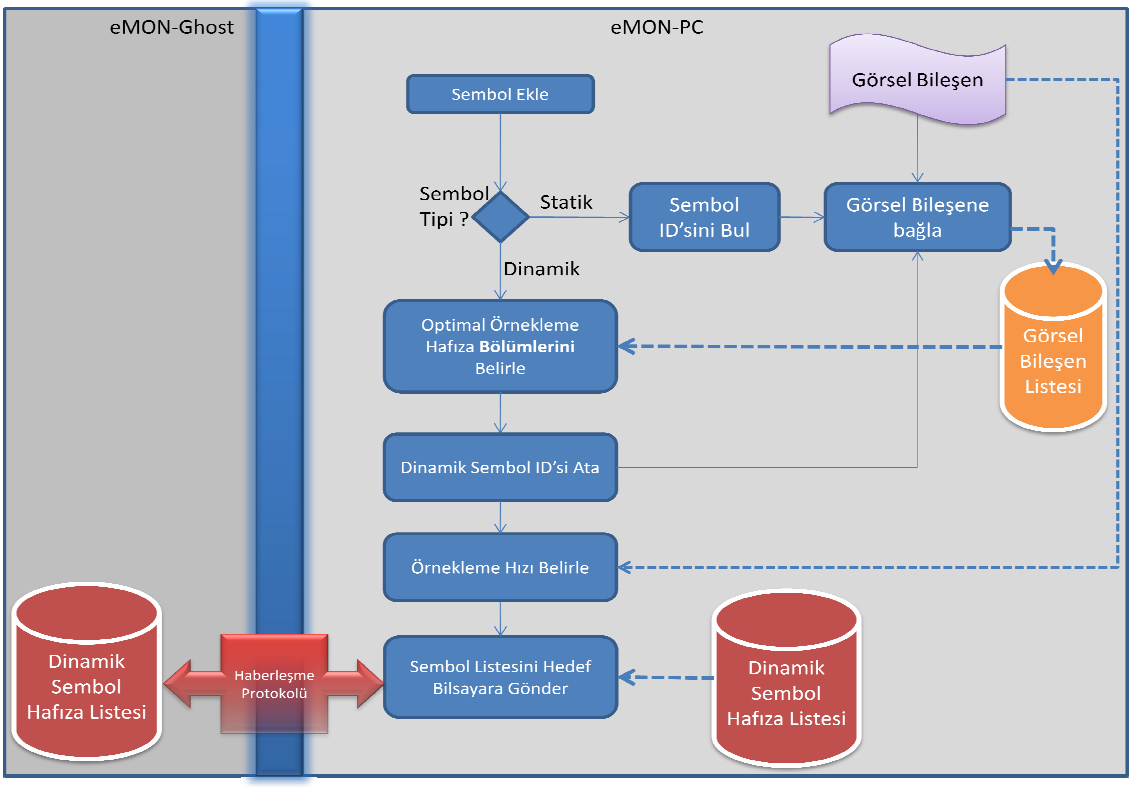
3.3.3.1 Sembol Ekleme

Dinamik ve statik sembollerinin parametre yöneticisi tarafından ele alınışı farklılık göstermektedir. Dinamik sembollerin adres ve boyut bilgilerinin hedef bilgisayara eMON-Ghost aracılığı ile kayıt ettirilmesi gerekmektedir. Kayıt işlemi yapılırken dinamik semboller en uygun şekilde gruplanarak hafızada sürekli olması sağlanır. Örneğin a,b,c sembolleri örneklenirken a ve b sürekli olduğu için adres alanı birleştirilerek 0x0-0xC adres aralığı için bir adet probe paketi; c sembolü içinse 0x1F-0x24 adres aralığı için ayrı bir probe paketi oluşturulur. Dinamik ve statik sembol ekleme akış diyagramı Şekil 5’de verilmiştir.

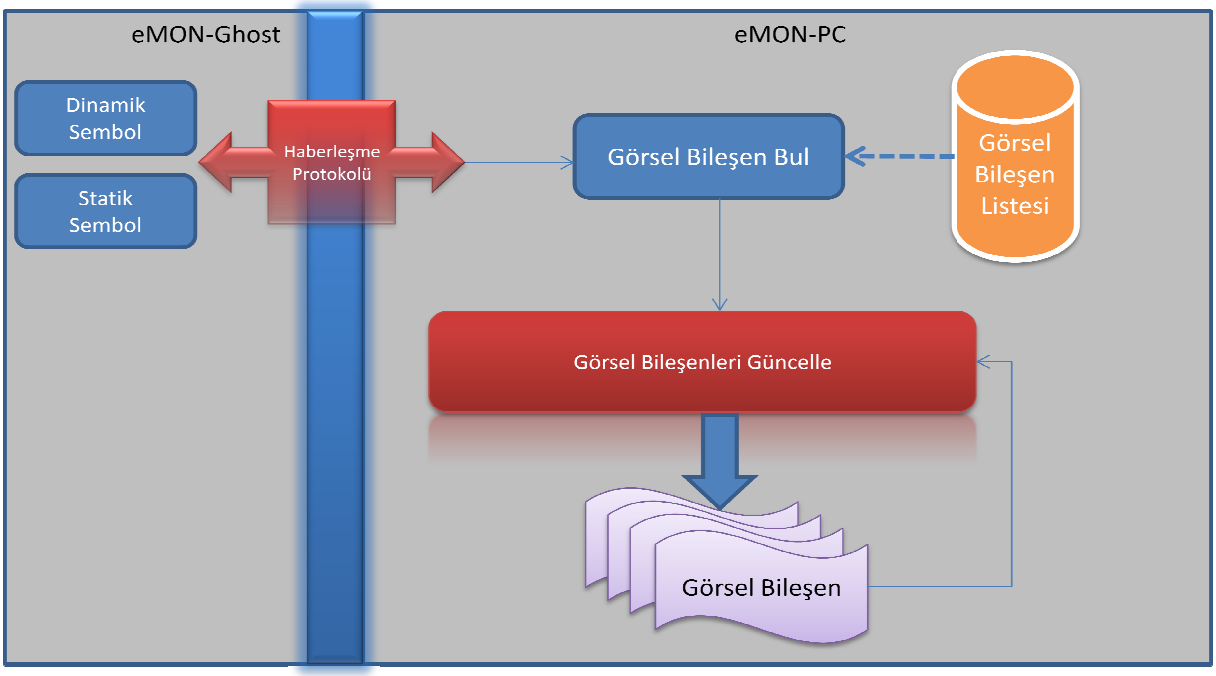
a: adres: 0x0 boyut: 10
b: adres 0xB boyut: 1
c: adres 0x1F boyut: 4

3.3.3.2 Sembol Güncelleme

Hedef bilgisayardan eMON-Ghost yazılımının gönderdiği sembol verilerinin işlenmesi Şekil 6’de verilmiştir. Semboller monitör tanımlama ID’lerine göre görsel bileşenlerle eşlendiğinden, gelen ID’ye bağlanmış tüm görsel bileşenler güncellenir. Bazı bileşenler diğer bileşenlerle port bağlantısı yapabildiğinden, bileşen çıktı portlarına bağlı olan diğer bileşenlerin güncellenmesi de gerçekleştirilir.



Şekil 5 Sembol Ekleme



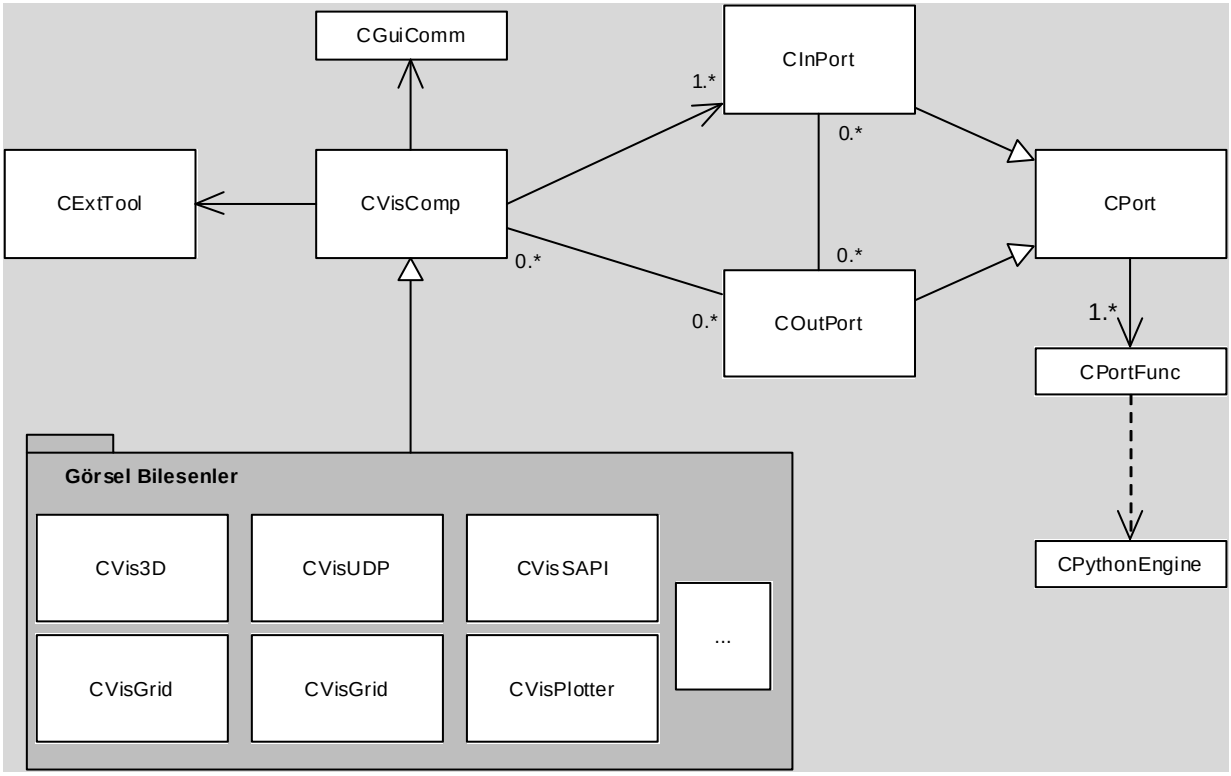
Şekil 6 Görsel Bileşen Güncelleme

3.4.4 Görsel Bileşen Mimarisi

Görsel bileşen sınıf yapısı Şekil 7'deki UML diyagramında verilmiştir. Görsel bileşenler CVisComp tipindeki taban sınıftan türemiştir. CVisComp sınıfı bileşen tanımlama, temel GUI fonksiyonları (pencere yönetimi), portlar arasında veri transferi, veri ara-bellekleme (Ing. buffering) gibi temel işlevleri gerçekleştirmektedir.

Görsel bileşen giriş/çıkış işlevlerini yönetmek için CPort sınıfından türeme CInPort ve COutPort sınıflarını kullanmaktadır. Giriş çıkış portları verileri üzerinde işlem yapabilmek için CPortFunc sınıfı mevcuttur. Bu sınıf kullanılarak portlar üzerinde temel operasyonlar (sin, cos, sqrt, fark,..) uygulanabileceği gibi karmaşık betik tabanlı operasyonlar da yapılabilmektedir.

CExtTool sınıfı ise herhangi bir 3. parti uygulamayı eMON-PC kullanıcı arayüzü içerisinde çalıştırmak için tasarlanmıştır. Bu sınıf sayesinde birden fazla uygulama daha kompakt bir yapı içerisinde kullanıcıya sunulabilmektedir.



Şekil 7 Görsel bileşen sınıf yapısı

3.5 eMON-Ghost

eMON-Ghost yazılımı hedef bilgisayarda çalışan C/C++ temelli basit bir gömülü yazılımdır. Bu yazılım işlevleri aşağıda sıralanmıştır:

- Dinamik sembol listesini yönetmek
- Dinamik sembol örneklemesini gerçekleştirmek
- Statik sembol gönderimini yapmak
- Sembol örnekleme ve gönderim istatistiklerini tutmak

Bu yazılımın ana fonksiyonu gerçek zamanlı işletim sistemi içinde bir ödevde(task/thread) ya da sabit bir frekanstaki herhangi bir fonksiyon içerisinde çağrılabilir. eMon-Ghost yazılımı sembol listesinin büyüklüğü ile doğru orantılı işlemci zamanı kullanımına sahiptir. Sembol listelerinin büyüklüğü artıkça, gerçek zamanlı

operasyonun bozulmaması için, işletim sisteminde bir ödev içerisinde çağrılması daha uygundur. Ödev önceliğinin yüksekliği bu yazılımın zaman kritik işlevlerle ne kadar etkileşeceğinin bir göstergesi olacaktır.

3.5 Sonuç

Bu çalışmada gömülü sistemlerin gözlemlenmesine ilişkin etkin bir yazılım mimarisi geliştirilmiştir. Dinamik sembol yönetimi kullanılarak ELF/DWARF tipindeki çalıştırılabilir kod içerisindeki herhangi bir sembol, çalışma zamanında hedef bilgisayardan örneklenerek gözlemlenebilmektedir. Statik sembol yönetim ile hedef yazılımdan dışarıya aktarılan verilerin XML tabanlı tanımlamaları gerçekleştirilerek eMON-PC yazılımı ile etkin görselleştirilmesi sağlanmıştır. eMON yazılım altyapısı kullanımı, yazılım geliştirme sırasında ve sonrasında hata ayıklama için harcanan zamanı kısalttığı görülmüştür. Bunun yanında eMON-PC yazılımının dünyadaki benzerlerinden farklı olarak XML/Python betik tabanlı olması ve görsel bileşenlerde toplanan veriler üzerinde işlem yapabilmeyi olanaklı kılmaktadır.

Kaynakça

[1] Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification, TIS Committee, 1995

[2] http://sourceware.org/gdb/current/onlinedocs/stabs.html

[3] “DWARF Debugging Information Format Version 3”, DWARF Standards Committee

[4] Eager, M. “Introduction to the DWARF Debugging Format”, Eager Consulting February, 2007

[5] Robbins,J. “Debugging Microsoft .NET 2.0 Applications”, Microsoft Press 2007

[6] Darling, D. “The Impact of DWARF on TI Object Files”, Texas Instrument Application Report SPRAAB5– August 2005

[7] http://micrium.com/page/products/tools/probe

[8] http://xidml.org/home

[9] “Python/C API Reference Manual”, http://docs.python.org/c-api/

Model Güdümlü Geliştirme ile Gömülü Kaynakların Yönetimi

Can Öz[1], Yasemin Topaloğlu[2]
[1,2] Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, 35100, Bornova, İzmir
[1] can.oz.mail@gmail.com, [2] yasemin.topaloglu@ege.edu.tr

Özet. Günlük yaşam içerisinde ve iş hayatında kullanmakta olduğumuz cihazların sayısı gün geçtikçe artmaktadır. Bu araçlar iletişim, bilgi edinme gibi isteklerle kullanılmasının yanında bazı işleri otomatik biçimde sonlandırmak isteğiyle de kullanılmaktadır. Bu araçların yeteneklerinin ve işlem kapasitelerinin yükselmesi kendi işlevselliklerini artırarak onların ortak biçimde yönetilmesini bir problem olarak ortaya çıkarır. Aynı işi yapan birden fazla kaynağın yönetilmesi, kaynaklar değişse bile ortak bir model üzerinden değerlendirilebilir. Bu çalışma içerisinde bu karmaşıklıkla baş edebilmek için güncel yazılım geliştirme tekniklerinden biri olan model güdümlü yazılım geliştirme alanından faydalanılacaktır. Üretilen Kaynak Yönetimi Metamodeli, beraber çalıştırılan kaynakların yönetilmesini sağlayan bir yönetici oluşturmayı hedeflemektedir. Model güdümlü geliştirmenin esaslarından biri olan model dönüşümü ile gerekli kurallar tanımlanarak kaynak yönetimi problemine uygun biçimde kaynak yöneticisi kodunun üretilmesi hedeflenmektedir.

Anahtar Kelimeler. Gömülü Sistemler, Kaynak Yönetimi, Kaynak, MDD, Model Güdümlü Yazılım Geliştirme

1 Giriş

Gömülü araçlar günümüzde birçok hizmet verilebilecek şekilde hazırlanmaktadır. Veriyi paylaşma, veriyi yönlendirme gibi yeteneklerin temelinde veriyi işleyebilme özelliği önem göstermektedir. Araçlar, yüklendikleri hizmet türüne göre onlara atanan işleri otomatik olarak tamamlayacaklardır. Ancak, onlara yönlendirilecek işlerin belli bir incelemeden geçirilmesi ve uygun zamanda kendilerine yönlendirilmesi gerekir.

Bu çalışmada, ortak özelliklere sahip gömülü araçların yönetilmesi için model güdümlü yazılım geliştirme teknikleri kullanılmıştır. Problemi soyutlaştırarak çözüm kümesinden problem kümesine yaklaşmak ve verimliliği arttırmak için model güdümlü yaklaşımların kullanılması tercih edilmiştir. Böylece çalışma sonunda, gömülü kaynakların yönetimi için bir metamodel tanımlaması yapılacaktır.

Hazırlanan çalışma, benzer yeteneği olan cihazların ortak biçimde yönetilmesi problemi üzerinde durmaktadır. Sistem içerisinde kaynak ve işler tanımlanacaktır. Sisteme gelen işin uygun bir kaynağa yönlendirilebilmesi için kaynağın ve işin özelliklerinin belirtilmesi gerekmektedir. Hazırlanan Kaynak Yönetimi Metamodeli, bu isteği karşılamaktadır. Çalışma sonunda kaynak yöneticisine ait Java dilinde kaynak koda sahip bir uygulamanın oluşturulması hedeflenmektedir.

Çalışmanın içeriği şu şekilde hazırlanmıştır. Model güdümlü yazılım geliştirme hakkında açıklamalar ikinci bölümde yapılmıştır. Bildirinin üçüncü bölümünde gömülü

sistemler ve gömülü sistemlerde yazılım geliştirme üzerine kısa açıklamalar bulunmaktadır. Dördüncü bölümde model güdümlü geliştirme ve kaynak yönetimi üzerine incelenen çalışmaların özetleri vardır. Beşinci bölümde kaynak yönetimi içerisinde bulunan kavramlar tanımlanacaktır. Altıncı bölümde metamodeli kullanan örnekler incelenecek ve sonraki bölümde alanla ilgili model dönüşümü hakkında bilgi verilecektir.

2 Model Güdümlü Geliştirme

Günümüzde yazılım geliştirme, geleneksel kodlama süreçlerinden model güdümlü yazılım geliştirmeye doğru yer değiştirmektedir(MGG). Modeller genelde karmaşık sistemleri daha esnek biçimde temsil etmek için kullanılmaktadır [1]. Amaç kod yazmaktan çözüm modelleri oluşturmaya taşınmıştır. Bu sayede hem müşteriler hem de geliştiriciler için ortak bir anlayış noktası oluşturulur. Model oluşturulduktan sonra problem alanından(*domain*) anlaşılanlar, çalışan, daha güvenilir koda, daha hızlı biçimde dönüştürebilecektir. Diğer geleneksel mühendislik disiplinlerinde olduğu gibi, modelleme ile gerçekleştirilen soyutlama, karmaşık problemlerin daha kolay anlaşılmasına ve olası çözümlere daha rahat ulaşılmasına yardım etmektedir. Model güdümlü yazılım geliştirme, yapısal programlamadan başlayıp nesne tabanlı programlamaya kadar yazılım geliştirme içerisinde yer almaktadır. Desteklenen soyutlama seviyesi, kullanılan programlama dilinin yeteneğine göre sınırlanmaktadır [2].

Belli bir alana özgü modellemenin yapılabilmesi için özel bir modelleme dilinin kullanımı gereklidir. Oluşturulan metamodel, sistemdeki istenmeyen detayları yok sayar ve sadece amaca uygun kısımların filtrelenerek incelenmesini sağlar. Diğer bir deyişle metamodelleme, modellerin oluşturulması için gerekli varlıkların tanımlamasını içeren soyut bir sınıfa benzemektedir. Metamodellerin tanımlanması da metametamodel ile yapılmaktadır. Model, metamodel ve metametamodel seviyeleri OMG(Object Management Group) tarafından belirlenmiştir. Modelleme katmanları MOF(Meta Object Facility) isimli metamodel kavramıyla tanımlanır [3].

Model güdümlü mimari, sistem işlevselliğini alana özgü dil (DSL) ve platform bağımsız model (PIM) kullanarak tanımlar. Alana özel dil, üzerinde çalışılacak konuya ait özel bir çözümün oluşturulması için kullanılmaktadır. Böylece bakımı kolay, güvenilir, taşınabilir ve yeniden kullanılabilir sistemler modellenebilir. PIM, gerçekleştirim sırasında kullanılacak herhangi bir teknolojik platformadan bağımsız olarak problemi çözmek için oluşturulan modeldir. PIM' e örnek olarak UML ile hazırlanan modeller verilebilir. Oluşturulan model daha sonra üst düzey bir dildeki gerçekleştirime dönüştürülebilecektir.

3 Gömülü Sistemlerde Yazılım Geliştirme

Gömülü yazılım geliştirmenin giderek karmaşıklaşmasında ortaya çıkan en önemli zorluklar; ürün kalitesi, pazara yetiştirilmesi ve düşük maliyet hedefleridir. Son yıllarda ortaya çıkan MGG (Model Güdümlü Geliştirme), gelecek vaat eden yaklaşımlara sahiptir. Geliştiriciler, doğrudan programlama dilleri kullanarak yazılım geliştirmek yerine, yazılım sistemlerini modellemektedir. Böylece daha anlaşılır, grafiksel işaretlere sahip doğal programlama dillerinden daha soyut seviyede otomatik kod üreten modeller

oluşturulabilecektir. Gömülü sistemlerde artan karmaşıklığın yönetilebilmesi elle kod üretmenin yerini modellemenin almasıyla sağlanabilir.

Diğer yazılım geliştirme süreçlerinde olduğu gibi gömülü sistemler içerisinde de yazılım geliştirme süreçlerini takip etmek gerekmektedir. Yinelemeli bir süreç olan UML tabanlı ROPES(Rapid Object-Oriented Process for Embedded Systems) temel olarak spiral süreç modeline benzemektedir [4]. Yinelemeli model güdümlü geliştirme süreci, gömülü sistemin geliştirme yaşam döngüsü içerisinde her fazı kapsamalıdır. Bu süreç sayesinde geliştiriciler daha kritik parçaları önce yapmak şartıyla her adımda kısmi gerçekleştirimler yapabilecektir.

4 İlgili Çalışmalar

Yazılım geliştirme içerisinde MGG teknikleri kullanan birçok örnek vardır. İncelenen çalışmalar ile MGG tekniklerinin nasıl kullanılacağı ve modelleme adımlarının nasıl oluşturulacağı gözlemlenmiştir. Gömülü sistemlerin modellenmesi için hazırlanmış örnek çalışmalar ile Kaynak Yönetimi probleminde dikkat edilmesi gereken noktalar belirlenmiştir.

4.1 Model Güdümlü Yazılım Geliştirme Tekniğini Kullanan Çalışmalar

Tahta Oyunları için Model Güdümlü Yazılım Geliştirme bildirisinde[5] tahta oyunları kümesi için metamodel tabanlı bir yaklaşım önerilmektedir. Tahta oyunları yapay zeka karmaşıklığına ve gerçekleştirilmesi zor grafiksel arayüzlere ihtiyaç duymadığından model tabanlı yazılım geliştirme süreçlerinin uygulanması için güzel bir örnek oluşturmaktadır. Öncelikle GameEngine, Player ve Rules kavramlarını tanımlayan metamodel tanıtılmaktadır, arkasından tahta oyunu için gerekli olan DSL söz dizimi soyut ve somut söz dizimi ile verilmektedir. Metamodel iki örnek oyunun tahta oyunları metamodelinden türetilerek sunulmasıyla tamamlanmaktadır.

Acil Durum Kaynak Yönetimi ve Koordinasyonu bildirisinde[6] acil durum bilgi sisteminin alt kümesi olan kaynak yönetiminin kullanılması için alana özel dil yaratılması hedeflenmektedir. İlk olarak alan çözümlemesi yapılmaktadır. Alan çözümlemesinin ardından, soyut söz dizimi, somut söz dizimi ve statik anlamsallığı içeren metamodelleme çalışması yapılır. Acil durum yönetiminin esas dayandığı nokta koordinasyondur. Başarılı bir yönetim için çeşitli seviyelerde ve aynı anda koordinasyonun bulunması gerekmektedir. Her seviyede koordinasyon değişik olabilir. Yerine göre bir kurumun tek başına cevap vermesi gerektiği gibi bazı durumlarda birçok kurumun beraber çalışması gerekebilir. Çalışılan örnek, sanayi standartı olan WS-BPEL metamodeli ve EDXL-RM kaynak yönetimi özellikleri temel alınarak hazırlanmıştır.

4.2 Model Güdümlü Gömülü Yazılım Geliştirme Akışı (Model based Embedded Software Development Flow)

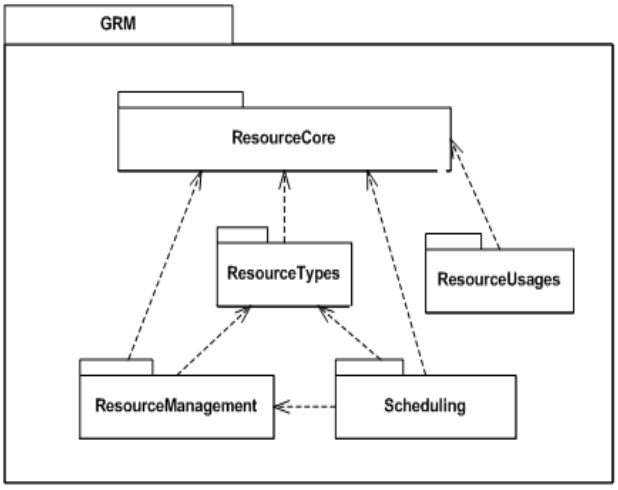
Gömülü yazılım geliştirme, değişen gereksinimler, karmaşık sistem mimarileri ve evrimleşen teknolojik platformlar gibi sorunlarla yüzleşmektedir. İncelenen çalışma[7] gömülü yazılım geliştirme konusunda UML tabanlı bir metodoloji önermektedir .

Hedeflenen tasarım iki temel faz içermektedir. İlki, xtUML(excecutable UML) kullanılarak yazılımın modellenmesi ve ikincisi modelin doğrulanmasından sonra modelden gömülü yazılım için kodun üretilmesidir. Yaklaşımın takip edilmesi, geliştirme süreci sırasında tanımlama hatalarının fark edilmesine ve model içerisindeki çeşitli kısımların tekrar kullanılmasına yardım etmektedir. Üç esas anahtar süreçten oluşmaktadır. İlk süreç otomatik kod geliştirme amacı için gömülü yazılım modülünün modellenmesini içerir. İkinci süreç, test aşamasının yaratılması için ayrıca modelleme görevini içermektedir. Üçüncü süreç içerisinde, yaratılan kodun yaratılan test ile otomatik olarak test edilmesini içermektedir.

4.3 MARTE(Modeling and Analysis of Real-Time and Embedded Systems)

MARTE eş zamanlı ve gömülü sistemlerin model güdümlü tanımlamaları için alt yapıları sahiptir. Modelleme bölümü, sistemlerin gerçek zamanlı ve gömülü karakteristiklerinin detaylı tasarımının yapılabilmesine destek olmaktadır. MARTE özellikle performans ve zamanlanabilirlik çözümlemesine odaklanmaktadır. Fakat yine de nitelik çözümlemesi için genel bir çalışma çerçevesi tanımlamıştır [8].

MARTE, dört temel direk üzerine kurulmuştur. Bunlar; QoS-aware Modeling, Architecture Modeling, Platform-based Modelling, Model-based QoS Analysis olarak adlandırılır. Platform Temelli Modelleme paketi içerisine Kaynak ve Kaynak Servislerini de dahil etmiştir. GRM(Generic Resource Modelling) adını verdiği paket içerisinde gerçek zamanlı gömülü uygulamaların çalıştırılması için gereken platformun modellenmesinde ihtiyaç duyulan kavramları tanımlar. MARTE'nin GRM paketi içerisinde orta nokta da *kaynak* kavramı bulunmaktadır. *Kaynak* bir davranış ile donatılmış, diğer bir deyişle bir veya birden fazla servis sunan bir sınıflandırıcıdır. *ResourceService* ise bir davranışı göstermektedir. *ResoruceCore* paketinde temel elemanlar ve onların ilişkileri belirtilir. *ResoruceTypes* paketi kaynakların asli tiplerini ve sağlamakta oldukları servisleri tanımlar. *ResourceManagement* paketi, yönetim kaynaklarını ve onların özelliklerini tanımlar [9]. Şekil 1 GRM paketinin mimarisini göstermektedir.



Şekil 1. GRM Paketinin Mimarisi (UML Profile for MARTE, V1.0)

5 Kaynak Yönetim Modeli

Kaynak Yönetim Modelinin kullanılacağı sistem içerisinde kaynaklar ve işlemler temel noktada bulunmaktadır. MARTE gömülü sistemlerde gerçek zamanlı yazılımların modellenmesi için bir UML profile sunarken Kaynak Yönetimi Metamodeli kaynağı oluşturan araçların kendi işlevselliklerinin yönetilmesini ortak bir model üzerinden yürütebilmeyi hedeflemektedir. Çalışma gömülü kaynakların yönetilmesi üzerinde durmaktadır çünkü bu kaynaklar genelde tek bir servis verme yeteneğine sahiptirler ve dışarıdan gelen işlem tiplerine göre doğru biçimde seçilmeleri gerekmektedir. Kaynağı oluşturan araçlar temelde aynı şekilde çalışıyor olsa da kaynağın tamamlaması gereken iş her zaman sabit bir önemde olmayabilir. Anlık gelen isteğin, o an hangi kaynağa yönlendirileceği takip edilebilmelidir ve aktif kullanılan kaynağın yeni bir istek tarafından kullandırılmaması gerekmektedir. Aynı şekilde isteği tamamlayan kaynağın yeni bir istek alabilecek şekilde bekleme durumuna döndürülmesi gerekir.

Kaynağın bir istek süresince durumları; hazırda bekleme, kullanımda ve bozuk biçiminde olabilir. Durumu her zaman değişebilecek bir kaynak için hazırda bekleme ve bozuk durumları belli sürelerle kontrol edilerek takip edilebilir. Sistemden gelen bir istek için kaynak ihtiyacı durumu hazırda bekleme olan kaynaklar tarafından karşılanacaktır.

5.1 Kaynak Yönetim Metamodelinin Elemanları

Kaynak Yönetim Metamodelinde yedi temel eleman tanımlanmıştır. Bunlar; Kaynak Yöneticisi, Kaynak, Kaynak Tipi, İşlem, İşlem Tipi, Durum, Durum İzleme olarak adlandırılmış olup, aşağıda kısaca açıklanmıştır:

Kaynak Yöneticisi: Sistemden sorumlu olup kaynak ve iş yönetimini yapan elemandır. İş, elde tutulan kaynaklara göre paylaştırılacaktır. Gelen işin tipine göre belirli kaynak kullanımı gerekebilir. Kaynağın durumu; işi yapmadan önce, işi yapma sırasında ve iş tamamlandıktan sonra takip edilebilmelidir. İşlem sırasında kullanılan kaynak, yeni gelen iş için kullandırılmamalıdır.

Kaynak: Bir servis verme yeteneğine sahip elemandır. Kaynak Yöneticisine gelen işler eldeki kaynaklara göre dağıtılmaktadır. Kaynağın durum bilgisi bulunmalıdır. Bazı kaynaklar işlem tipine göre ayrılabilir.

Kaynak Tipi: Kaynağın verdiği servisin hangi işlem ile eşleştirildiği bilgisini ve kaynak tipinin tanımını tutmaktadır.

İşlem: Kaynakların ne amaçla kullanılacağını belirtmektedir. Her işin kendine özel işlem tipi bulunmaktadır.

İşlem Tipi: Yapılacak işin hangi önceliğe sahip olduğunu belirten elemandır. İşlem tiplerine göre kaynaklar ayrılabilir. Önceden meşgul olan bir kaynak daha öncelikli bir işe atanabilir. İşin kaynaklara paylaştırılabileceği (paralel çalıştırılabileceği) durumlar varsa bazı kaynaklar paralel çalışabilecek şekilde ayarlanabilir.

Durum: Sistemdeki kaynakların sahip olabileceği durumları belirtir. *Kaynak boşta*, *kaynak kullanımda*, *kaynak arızalı* örnek olarak verilebilir.

Durum İzleme: Herhangi bir zaman içerisinde kullanılan kaynakların hareketlerini izlemektedir. Kaynağın kullanım süresi, yapılan iş tipi, sistemde bulunan diğer

kaynakların durumu takip edilebilir. Gereken durumlarda o an işlenen işin yapılma süresi gibi istatistik bilgileri de hazırlanabilecektir.

5.2 Kaynak Yönetimi Modellemesinde Kullanılan Araçlar

Modellemenin gerçekleştirilebilmesi için bazı araçlara ihtiyaç vardır. Bu araçlar sayesinde metamodellerin tanımlanması ve birbirleri arasında dönüştürülmesi gerçekleştirilebilir. Eclipse [10], Java diliyle hazırlanmış bütünleşik geliştirme ortamıdır. Birçok eklentiyle uygulama geliştirmeye verdiği desteğin yanında Model Güdümlü Yazılım Geliştirme konusunda da çeşitli eklentilere sahiptir.

Eclipse içerisinde MOF' un tanımladığı modelleme için gerekli olan yapılar EMF (Eclipse Modelling Framework) içerisinde bulunmaktadır. Diğer bir deyişle MOF tarafından soyut biçimde ifade edilen varlıklar Eclipse içerisinde EMF ile Java programlama dilinde gerçekleştirilmiştir. EMF, Ecore diye adlandırılan meta dile sahiptir. Bu dilin sahip olduğu yapılar, MOF 2.0 ile beraber MOF' dan ayrılarak ortaya çıkan bir model olan EMOF (Essential MOF) ile bazı benzerliklere sahiptir. EMF iki önemli yapıdan oluşmaktadır: çekirdek çerçeve ve EMF.Edit. Çekirdek çerçeve, verilen bir model için gerekli olan esas üretimi ve çalışma anı desteği için Java gerçekleştirim sınıflarını sağlamaktadır. EMF.Edit ise çekirdek çerçeveyi desteklemek için gerekli olan modellerin düzenlenmesi ve ayarlanması işlerine yardımcı olmaktadır.

5.3 Soyut Söz Dizimi

Şekil 2' de Kaynak Yönetimi metamodelinin soyut sözdizimi belirtilmiştir. Metamodel, Eclipse geliştirme ortamına modelleme yeteneğini katmak için kullanan EMF içerisinde bulunan ECORE ile hazırlanmıştır.

5.4 Somut Söz Dizimi

Kaynak yönetimi metamodeli aynı işi yapan araçlara yönlendirilecek işlerin sıralanması ve uygun olan kaynağın kullanılmasını sağlamaya yarayacaktır. Metamodele göre tanımlanacak model içerisinde iki temel elemandan biri *Kaynak* diğeri de bu kaynağın servis verdiği *İşlemdir*.

Yönetici ::= (Kaynak+, İşlem+);

Kaynak yöneticisi kaynakları ve işlemleri yönetmektedir. Bu bilgi ile, kendisi üzerinden yönlendirilen işe hangi kaynağın uygun olacağı bulunabilir. Bir kaynağı tanımlamak için, kaynak adı, tipi ve hangi durumlara sahip olacağı belirtilmelidir. Her kaynak sadece bir servis verebilir.

Kaynak ::= (kaynak adı, kaynak tipi, durumu);

kaynak adı ::= ad;

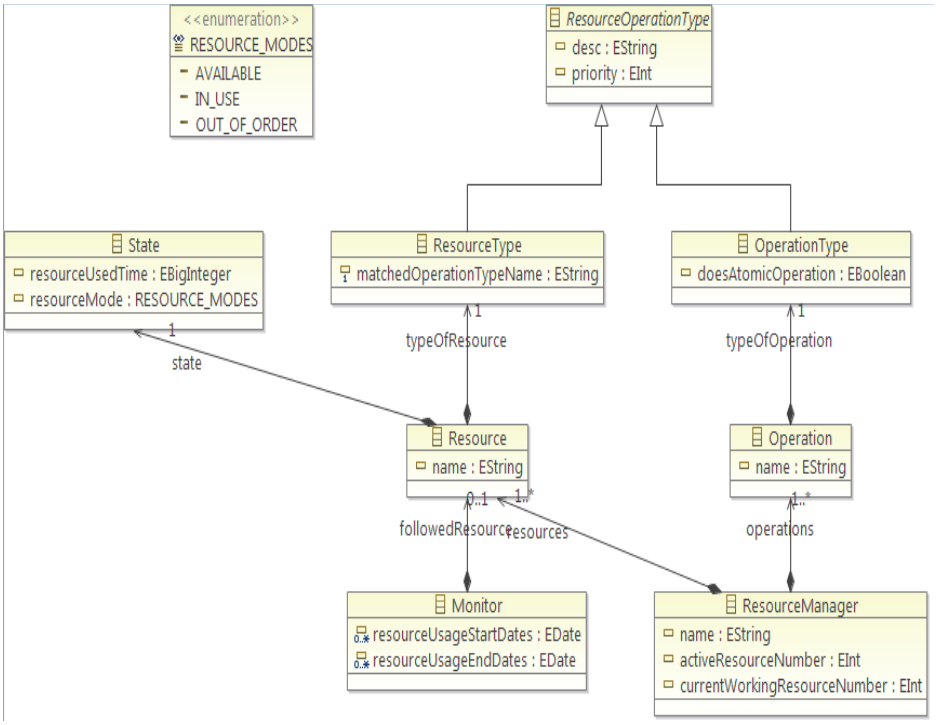
kaynak tipi ::= (kaynak tipi adı, eşlendiği işlem tipi adı+,
kaynak tipi tanımı);

kaynak tipi adı ::= ad;

eşlendiği işlem tipi adı ::= işlem tipi adı ;

İşlem	::=	işlem tipi;
işlem tipi	::=	işlem tipi adı, öncelik, [bölünebilir iş özelliği];
işlem tipi adı	::=	ad;
durum	::=	'UYGUN' | 'KULLANIMDA' | 'BOZUK';
öncelik	::=	RAKAM;
ad	::=	HARFLER;
kaynak tipi tanım	::=	HARFLER;
RAKAM	::=	'0' | '1' | '2';
HARFLER	::=	('a' .. 'z' .. 'A' .. 'Z')+

Kaynağın vereceği işlem, işlem adı ve işlem tipine göre belirlenecektir. İşlem tipi, hangi duruma sahip olan kaynakta çalıştırılabileceğine göre eşlenebilmelidir. Öncelik türü ve bölünebilirlik özelliği ile bir kaynak üzerinde çalıştırılan işin bekletilmesi belirlenecektir.



Şekil 2. Soyut söz dizimi

5.5 Statik Anlam

Bir metamodelin anlamı o modelin iyi tanımlanmış kurallarını anlatmaktadır. Bu kurallar ile metamodelin hangi kısıtlamalara sahip olacağı, modelin nasıl şekilleneceği ve doğrulanacağı tanımlanacaktır. OCL(Object Constraint Language) veya openArchitectureWare içerisinde kullanılan Check (OCL benzeri) dili hem model hem de metamodel seviyelerinde bu kuralların yazılabilmesine yardımcı olmaktadır.

Aşağıda Kaynak Yönetim Metamodeline ait kurallardan bazıları Check dili ile ifade edilmiştir. Buna göre, Kaynak Yöneticinin en az bir tane çalışan Kaynak elemanı olmalıdır. Kaynak elemanı mutlaka isim bilgisine sahip olacaktır. Sistem içerisinde bulunan her kaynağın durum bilgisi belirtilmelidir. Her kaynak belli bir işlem tipini servis edecektir. Bu servisin isim bilgisi bulunacaktır.

```
Context ResourceManager ERROR "resource manager should have
more than one active resource number"
      activeResourceNumber > 0;

Context  Resource ERROR "zero resource name not allowed"
      name.length > 0 ;

context Resource ERROR "every resources' state should not
be null"
      state != null;

context Resource ERROR "every resource must have one
operation"
      operation != null && operationNumber == 1;

Context  Operation ERROR "zero  operation name not allowed"
      name.length > 0;
```

6 Örnek Modeller

6.1 Yazıcı Uygulaması

Hazırlanan metamodel yazıcılara iş dağıtımı yapan bir uygulamayı geliştirmek için kullanılacaktır. Her yazıcının gerçekleştirebildiği işlem için belirli işlem tipleri bulunmaktadır. Örneğin yazıcılar renkli yazıcı ve renksiz yazıcı olarak ikiye ayrılabilir. Gelen her işin belirli bir önceliği bulunmaktadır. Bu önceliğe göre eldeki kaynaklara iş atanması ve gerekirse bir kaynağın daha öncelikli yazdırma işi için boşa çıkartılması gerekebilir. Her yazıcının o anki durumları kontrol edilmektedir. Kağıdın bitmiş olması, kartuşun bitmiş olması, yazıcıya ulaşılamaması gibi durumlarda yazıcı devre dışı bırakılabilir. İstenilen bir zamanda, yazıcıların o anki durumuna ulaşılacaktır.

6.2 Akıllı Kart Okuma Uygulaması

Metamodelin kullanılabileceği diğer bir uygulama akıllı kart okuma cihazlarının yönetimidir. Bir bilgisayara seri porttan bağlı akıllı kart okuma cihazları, seri iletişim yoluyla aldıkları bilgileri ilgili sunuculara gönderip gelen cevabı onlara isteği yapan kart okuma yöneticisine ileteceklerdir. Akıllı kart okuma cihazlarına gelecek işlem tipi iki tanedir. Bu işlem tiplerine anında cevap verme ve beklemede adı verilmektedir. Akıllı kart okuma cihazları, bu işlem tiplerine göre isteğe bağlı biçimde atanabilirler. Örneğin, bu cihaza gelen işlem, anında işlem ise kart okuyucu doğrudan sunucuya bağlanır ve kendisine gelecek cevabı kart okuyucu yöneticisine iletir. Beklemeli işlemler belirli bir sayıya ulaştığında sırayla sunucuya iletilebilir. Bu işlem tipi için gelecek cevabın yöneticiye döndürülmesi gerekmemektedir. Kart okuma cihazları belirli işlem sayısına ulaştıklarında devre dışı kalabilirler. Bu sebeple yöneticinin, işlem yapmadan önce işlem yapacağı kart okuyucunun hazırda olup olmadığını kontrol etmesi gerekmektedir.

7 Model Dönüşümü

Model güdümlü yazılım geliştirme içinde dönüşüm kurallarını takip ederek girdi kısmında bir veya daha fazla kaynak model alınıp, çıktı kısmında bir veya daha fazla hedef model üretilmesi sürecine model dönüşümü adı verilmiştir [11].

Kaynak Yönetim Metamodeli ile modellenebilecek uygulamalar altıncı bölümde anlatılmıştır. Metamodel üzerinde oluşturulacak dönüşüm kuralları ile Java gerçekleştirim dilinde otomatik kaynak kodun üretilmesi hedeflenmektedir. Yazıcı uygulamasında yazıcılar kaynakları oluşturmaktadır. Her yazıcı renkli veya renksiz çıktı verme servisine sahiptir. Kaynak Yöneticisine gelen istekler ilgili yazıcıların o anki durumu kontrol edilerek uygun bir yazıcıya yönlendirilecektir.

Benzer bir model akıllı kart okuma uygulamasında da kullanılabilir. Akıllı kartlar, kendi işletim sistemlerine sahip, özel donanım ve yazılım özellikleriyle gelişmiş şifreleme hizmetleri sunan küçük bilgisayarlardır. Oluşturulan sistemde akıllı kartlar, güvenlik numarası üretme işinde kullanılacaktır. Ancak aynı anda gelen isteklerin uygun okuyucuya yönlendirilmesi gerekmektedir. Model içerisinde belirtilecek işlem tipine göre gelen istek uygun okuyucu tarafından karşılanacaktır.

Tablo 1. Örnek modellerin Kaynak Yönetimi Metamodeline göre eşlenmesi.

KAYNAK	Yazıcı	Kart Okuyucu
KAYNAK TİPİ	Renkli Yazıcı	Biriktirmeli cevap veren kart okuyucu
İŞLEM	Yüksek öncelik	Normal öncelik
İŞLEM TİPİ	Renkli	Beklemeli İş
DURUM	Kullanımda	Hazır

8 Sonuç

Yazılım geliştirme süreci, Model Güdümlü Yazılım Geliştirme ile çözüm kümesinden problem kümesine taşınmaktadır. Bu çalışmada kaynak yönetimi alanına ait bir problemin modellenmesi hedeflenmiştir. Tanımlanan metamodel ile kaynakların yönetilmesi için gerekli elemanlar ve bu elemanların birbirleriyle olan ilişkileri bir model içerisinde tanımlanmıştır. Böylelikle gömülü sistemler alanında çalışan yazılım geliştiriciler farklı uygulamalarda kullanabilecekleri bir kaynak yönetim modeline sahip olacaklardır. Model tamamlandıktan sonra yapılması hedeflenen, modellenen probleme ait kaynak kodunun oluşturulmasıdır. Bunun için gerekli olan dönüşümlerin tanımlanmasıyla gömülü kaynakları yönetecek uygulama geliştiricilerin üretkenliği artacak ve yazılım geliştirme soyutlama seviyesi yükselecektir.

Hazırlanan Kaynak Yönetimi Metamodeli temel noktada Kaynak elemanından oluşmaktadır. Kaynak elemanının özellikleri MARTE GRM paketinde bulunan tanımlamalara göre şekillendirilmiştir. Ancak MARTE'nin belirlediği kavramlar kaynakları yönetmek için yeterli değildir. Bu tanımlamalar ile kaynakların kapasiteleri, servisleri ve iletişim kanalları gösterilmektedir. Kaynaklar, gömülü cihazlarda bulunan hafıza bölgeleri olarak düşünülmüş, performans ve zamanlanabilirlik durumları göz önünde bulunarak tasarlanmıştır. Oysa ki Kaynak Yönetim Metamodeli, birden fazla kaynağın benzer işleri otomatik olarak tamamlayabilmesi için üretilmiştir. Metamodelde, Kaynak ve İşlem elemanları ön planda tutulmaktadır. Kaynak Tipi ve İşlem Tipi elemanları ile hangi kaynağın hangi işi yapabileceği işaretlenmektedir.

Modelleme aşamasında Eclipse geliştirme ortamı tercih edilmiştir. Çünkü Eclipse, sağladığı diyagramlar ile ve birbirleriyle uyumlu eklentileriyle modelleme sürecini kolaylaştırmaktadır. Kaynak yönetimi soyut söz dimi EMF içerisinde tanımlanmış olan ECORE ile hazırlanmıştır. ECORE, XMI türünde saklandığı için ECORE ile hazırlanan bir model başka XMI destekleyen bir platforma kolaylıkla taşınmaktadır.

Her ne kadar Eclipse modelleme sürecini basitleştirse de metamodel oluşturma, diğer bir deyişle M2 seviyesinde düşünme kolay bir işlem değildir. Metamodelleme sonrasında MGG' nin en önemli meydan okuması dönüştürme aşamasındadır. Dönüştürme, üst seviye bir modelden platforma özel bir modele dönüşümü içermektedir. Kaynak Yönetim Metamodeli oluşturulurken en çok modele ait soyut söz diziminin oluşturulması sırasında sıkıntı çekilmiştir. Dönüşüm çalışmalarıyla beraber modelden koda dönüşüm adımları tamamlanacaktır.

Kaynakça

1. Meservy, T.O. ve Fenstermacher, K.D.; “Transforming Software Development: An MDA Road Map”, IEEE, Vol. 38, No. 9 (2005), s. 52-58
2. Selic B.; “The Pragmatics of Model-Driven Development”, IEEE (Software), Vol. 20, No. 5, (2003), s. 19-25
3. Seidewitz E.; “What Model Means”, IEEE (Software), Vol. 20, No. 5. (2003), s. 26-32
4. Douglass B. P.; “Doing Hard Time: Developing Real-Time Systems using UML, Objects, Frameworks, and Patterns Reading”, MA: Addison-Wesley, 1999
5. Dogan Altunbay, Eser Çetinkaya, M.Gokhan Metin ; “ Model Driven Development of Board Games”, TMODELS 2009

6. Ilker Murat Karakas, Dogan Kaya Berktas, Eren Algan ; “Emergency Resource Management and Coordination”, TMODELS 2009

7. Kashif, H.; Mostafa, M.; Shokry, H.; Hammad, S.; “Model-Based Embedded Software Development Flow”, 4. International Design & Test Workshop (IDT), 2009, s. 1-4

8. The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, Haziran 6, 2010, http://www.omgmarte.org/

9. Khan A.M.; Mallet F.; Andre C; Simone R.; “Marte Timing Requirement and Spirit IP-XACT”, INRIA, (2009)

10. Mastering Eclipse Modeling Framework. Eclipse CON. 2005, http://www.eclipsecon.org/2005/presentations/EclipseCon2005_Tutorial28.pdf

11. Sendall S. ve Kozaczynski W.; “Model Transformation – the Heart and Soul of Model-Driven Software Development”, IEEE (Software), Vol. 20, No. 5. (2003), s. 42-45

Yazılım Ürün Aile Mühendisliği - IV

TADES:Komuta Kontrol Alanında bir Yazılım Ürün Hattı Çalışması

Ertuğrul Barak[1], Sezen Erdem[2], Hakan Yılmaz[3]

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

Özet. Yazılım Ürün Hattı belirli bir alanda, düşük maliyetle daha kaliteli yazılımları çok daha hızlı bir şekilde geliştirmeye yönelik ortaya çıkan bir yaklaşımdır. Bu yaklaşımın ortaya çıkışı pek çok çözülmesi gereken problemi de beraberinde getirmiş ve farklı akademik kuruluşlarda, bu problemlerin çözümüne yönelik farklı ekolleri meydana getirmiştir. Carnegie Melon Üniversitesi, Yazılım Mühendisliği Enstitüsü(CM SEI) de bu ekollerden birinin yaratıcısı olarak gösterilebilir.
Yazılım Ürün Hattı Yaklaşımının, özellikle otomotiv ve mobil telefon üreticisi firmalarında, başarılı bir şekilde uygulanması, diğer alanlarda, özellikle Savunma Sanayii alanında da uygulanmasına yönelik motivasyon kaynağı oluşturmuştur. Bu makalede ASELSAN'da Komuta Kontrol Yazılımları alanında bir Yazılım Ürün Hattı oluşturmak üzere yapılan TADES çalışması ve bu çalışmada elde edilen tecrübeler aktarılmaktadır. TADES çalışmasında CM SEI ekolü temel alınmakla birlikte yazılım geliştirme geçmiş tecrübeleri ve yazılım ürün hattı ile ilgili farklı ekollerin bir sentezi kullanılmaya çalışılmıştır.

1 Giriş

Günümüzde özellikle kişilerin ihtiyaçlarına göre uyarlanabilen yazılımlara, yazılım yoğun sistemlere ve hizmetlere olan yüksek talep yazılım dünyasında uyarlanabilir, ve yüksek kaliteli yazılımları hızlı bir şekilde geliştirme ve pazara sunma konusunda ciddi bir motivasyon oluşturmuştur. Uyarlanabilirlik, kalite ve hızlı geliştirme konularındaki rekabet, bu tip yazılımların geliştirilmesi sırasında uygulanacak yaklaşım konusunda farklı arayışlara neden olmuştur. Böylece, Yazılım Ürün Hattı Yaklaşımı ortaya çıkmıştır. Yazılım Ürün Hattı Yaklaşımında belli bir ürün ailesine mensup yazılımlar, bu ürün ailesinin ortaklıkları ve değişkenlikleri göz önünde bulundurularak, önceden oluşturulmuş ürün yapıtaşlarının bir araya getirilmesiyle oluşturulur. Bu bir araya getirme işlemi klasik yazılım geliştirme yönteminden daha çok bir yazılım entegrasyonuna benzer.

Bu yaklaşım ilk olarak mobil telefon ve otomotiv sektöründe başarıyla uygulanmış olması, diğer sektörlerde de yaklaşıma olan ilgiyi arttırmıştır. Bahsi geçen sektörlerden birisi de Savunma Sanayidir.

Ülkemizin önde gelen Savunma Sanayi kuruluşlarından ASELSAN da yazılım dünyasındaki bu gelişmeleri yakından takip etmektedir. Bu makalede ASELSAN'da

Teknik Ateş Destek Sistemleri alanında Komuta Kontrol Yazılımlarının geliştirilmesi için oluşturulan TADES yazılım ürün hattı çalışması anlatılmaktadır.

Bu makalede, ikinci bölümde Yazılım Ürün Hattı yaklaşımı hakkında özet bilgi verilmiş, üçüncü bölümde TADES'in problem alanı olan Teknik Ateş İdaresi hakkında bilgilendirme yapılmış, dördüncü ve beşinci bölümler TADES'in gerekçelerine ve TADES'in oluşturulması sürecine ayrılmıştır. Altıncı ve yedinci bölümlerde bu süreçte yaşanan tecrübeler ve kazanımlar aktarılmaya çalışılmıştır.

2 Yazılım Ürün Hattı

Yazılım Ürün Hattı, belirli bir pazarın özel ihtiyaçlarını karşılamak üzere, ortak ve yönetilen bir özellik grubunu destekleyen çekirdek bir varlık kümesinden, önceden tanımlanmış bir yöntem ile geliştirilen yazılım yoğun sistemlerdir[1].

Bir başka deyişle yazılım ürün hattı, benzerlikleri olan bir grup yazılım ürününün ortak bir mimariye dayandırılarak ve önceden geliştirilmiş ortak varlıkları kullanarak geliştirilmesi yaklaşımını ifade etmektedir. Bu yaklaşım, ürünlere ve yeniden kullanıma odaklı bir yaklaşımdır ve piyasanın durumuna ya da müşterinin isteklerine en hızlı şekilde tepki verebilmeyi öngörmektedir.

Yazılım Ürün Hattı yaklaşımı sanayide kullanılan üretim bantlarından esinlenmiştir. Bu yaklaşım temelde üç noktada fayda sağlamaktadır. Bu noktalardan birincisi yazılım geliştirme maliyetinin düşürülmesidir. Temelde bu yaklaşım yazılım ürünlerinin, ortak varlıkların doğrudan, değiştirilerek ya da ayarlanarak kullanılması yoluyla üretilmesi prensibine dayandığı için yeniden kullanılabilirliği üst düzeye çıkarmaktadır. Başlangıçta yeniden kullanılacak varlıkların oluşturulmasının belirli bir maliyeti olmasına karşın, bu ortak varlıkların kullanılmasıyla üretilen ürünlerin sayısı arttıkça yazılım geliştirmedeki genel maliyetin ciddi oranlarda düşmesi sağlanmaktadır.

Fayda sağlanılan ikinci nokta yazılım ürünlerinin geliştirme zamanının kısaltılması ve piyasaya çıkışlarının hızlı olmasıdır. Bu fayda da yine aynı prensipten kaynaklanmaktadır ve piyasaya çıkış süresindeki kısalma aynı alandaki rakiplerle rekabette avantaj sağlamaktadır.

Yaklaşımın üçüncü temel faydası da yazılım ürünlerinin kalitesinin artmasıdır. Ortaya çıkan ürünler daha önceden geliştirilen ortak varlıklara dayandığı için, bu varlıkların her yeni ürünle test ediliyor olması hataların önceden tespit edilme ihtimalini arttırmakta ve her bulunan ve düzeltilen hata sonrasında ürünlerin ve dolayısıyla yazılım ürün hattının kalitesi artmaktadır.

3 Teknik Ateş Destek Sistemleri

Ateş Desteği, muharebe sahasında ön saflarda yer alan birliklerin uzun menzilli obüs, top ve havan gibi silahların ateş gücünün kullanılmasıyla desteklenmesini ifade etmektedir. Teknik Ateş Desteği ise sağlanan bu desteğin teknik problemlerinin

çözülmesini kapsar. Bu nedenle Teknik Ateş Desteği temelde, yapılan atışların teknik esaslarının, kullanılan silah, mermi, barut gibi unsurların balistik özellikleri ve ortam koşullarının(meteoroloji) değerlendirilmesi ile belirlenmesi konusuyla ilgilenir. Teknik Ateş Destek Sistemleri yukarıda bahsi geçen alanda çözümler üreten, donanım ve yazılım unsurlarının bir araya getirilmesiyle oluşturulan sistemleri ifade etmektedir.

4 TADES: Teknik Ateş Destek Sistemleri Yazılım Ürün Hattı

Aselsan'da geliştirilen Ateş Destek Yazılımlarının geçmişi 2000 yılının öncesine uzanmaktadır. İlk başlangıcından itibaren bu yazılımlar teknik seviyeden başlayıp taktik seviyeye kadar uzanan bir yelpazede çeşitlilik göstermiştir. Bu alandaki yazılımlar farklı proje planları çerçevesinde farklı ekiplerce geliştirilmiş, zaman içerisinde bu alanda yazılım geliştiren personelde de büyük oranda değişim olmuştur. Bu etkenler dolayısıyla aynı alanda geliştirilen bu yazılımlar arasında yeniden kullanım potansiyeli çok yüksek olmasına rağmen, yeniden kullanım oranı istenen seviyede gerçekleştirilememiştir. Bu da yazılımların kalitesini etkilemekle beraber, idame, bakım, test ve eğitim gibi konularda sıkıntılar yaşanmasına neden olmuştur.

Bunun dışında yeni imzalanması beklenen yurt içi ve yurt dışı sözleşmeler ile bu alanda yeni yazılımlar geliştirme ihtiyacı ortaya çıkmıştır. Önceden sadece yurt içi müşteri göz önünde bulundurularak yapılan tasarımların yurt dışı ihtiyaçlarını karşılamada yetersiz kalacağı değerlendirilmiştir.

Bütün bunların dışında üst yönetim düzeyinde alınan kurumsal anlamdaki bir takım kararlar, Ateş Destek Alanında geliştirilecek yazılımlara ayrılabilecek işgücü kaynağının sınırlı bir seviyeye çekmiştir. Bütün bu koşullar değerlendirildiğinde Teknik Ateş Destek yazılımlarının geliştirilmesi, idame edilmesi ve bu sürecin yönetilmesi ile ilgili olarak farklı bir yaklaşımın uygulanması gerektiği değerlendirilmiş, ilgili idari yönetimin de desteğiyle TADES Yazılım Ürün Hattı çalışmasına başlanmıştır.

5 TADES Süreci

Yazılım ürün hattına geçiş, genelde üç adımda gerçekleştirilmektedir. Bu adımlar *Ürün Hattı Temel Varlıklarını Oluşturma, Temel Varlıkların İdamesi ve Ürün Geliştirme* ve *Ürün Hattının İdamesi ve Geliştirilmesi* olarak ifade edilmektedir.[1] TADES için de bu üç adımın uygun olduğu değerlendirilmiştir. Aşağıdaki bölümlerde bu adımlar altında TADES çalışmasında yürütülen faaliyetler özetlenmektedir.

5.1Ürün Hattı Temel Varlıklarını Oluşturma Aşaması

Yazılım Ürün Hattının oluşturulmasındaki ilk adım ürün hattı temel varlıklarının oluşturulması adımıdır. Bu adımın iki şekilde uygulanabildiği görülmektedir. "Büyük Patlama"(Big Bang) olarak adlandırılan birinci yaklaşımda son yazılım ürünü geliştirmeye odaklanmadan bütün kaynaklar temel ürün hattı varlıklarını geliştirmeye atanır. Uzunca bir süre bir yazılım ortaya çıkmaz ve sadece yeniden kullanım varlıkları veritabanının oluşturulmasına çalışılır. İkinci yaklaşımda ise temel varlık veritabanının geliştirilmesi bir yazılım geliştirme projesi sponsorluğunda gerçekleştirilir. Bu yaklaşıma da Yumuşak Geçiş (Smooth Transition) adı verilir.[2] TADES için ikinci yaklaşımın uygulanması bir zorunluluk olarak ortaya çıkmıştır.

TADES çalışmasında benimsenen CM SEI yaklaşımı, yazılım ürün hattı konusunda bir çerçeve tasarım sunmaktadır. CM SEI'nin yaklaşımında ilk etapta yazılım ürün hattı kurulması çalışması için öncelikle bir İşletme Konsepti dokümanı hazırlanması öngörülmektedir. Bu dokümanın içeriğinde neler olacağı ve dokümanın neleri adreslemesi gerektiği de ayrıca SEI'nin hazırladığı rehber niteliğindeki bir dokümanda belirtilmektedir.[3]

CM SEI Çerçeve tasarımında, yazılım ürün hattı kurma çalışmasına başlamadan önce çeşitli uygulama alanlarında belirli bir olgunluğa ulaşılmasının ve bu alanlara yönelik bir ön çalışma yapılmasının önemli olduğu belirtilmektedir. Bu uygulama alanlarının sayısı 29 adet olup bunlardan sadece 9 tanesi Yazılım Mühendisliği kapsamına girmekte geri kalan 20 alan ise ağırlıklı olarak Teknik ve İdari Yönetim kapsamında yer almaktadır. Bu da yazılım ürün hattı çalışmasının salt bir yazılım konusu olmadığını ve yönetimsel kısmının daha ağır bastığını göstermektedir.

TADES çalışmasına başlarken de ilk etapta CM SEI tarafından üzerinde önemle durulan yukarıda belirtilen konular dikkate alınmış ve çalışmalara bir TADES İşletme Konsepti dokümanı hazırlanmasıyla başlanmıştır. Bu dokümanda TADES yaklaşımına geçişin temel nedenleri belirtilmiş, öngörülen çalışmalar ve yaklaşıma geçişin adımları adreslenmeye çalışılmıştır. Hazırlanan bu doküman Aselsan içinde Teknik Ateş Destek Yazılımı geliştirme konusunda paydaş olan çeşitli fonksiyonel bölüm katılımcılarına sunulmuş, neden böyle bir yaklaşım değişikliğine gidilmesi gerektiği anlatılmıştır. Bu aşamadan sonra paydaşlar ile böyle bir uygulamaya geçilmesi konusunda fikir birliğine varılarak fiilen çalışmalara başlanmıştır.

Fiili çalışmaları planlama safhası ve uygulama safhası olarak ikiye bölmek mümkündür.Aşağıda bu iki safhada yapılan faaliyetlere daha detaylı bir şekilde değinilecektir;

Planlama Safhası.

TADES çalışması içerisinde planlama safhasında SEI tarafından belirtilen uygulama alanları üzerinde bir değerlendirme yapılmış ve hangi uygulama alanları üzerinde çalışmalar yapılması gerektiği bir karara bağlanmıştır. Bu karar doğrultusunda planlama aşamasında aşağıdaki çıktıların oluşturulmasına karar verilmiştir.

İş Durumu Dokümanı. Yazılım Ürün Hattı yaklaşımına geçişin temel sebeplerini içermektedir.

Eğitim Planı. Yazılım Ürün Hattı yaklaşımına geçiş öncesi, teknik ve alana yönelik eksikliklerini gidermek üzere personele verilecek eğitimlerin planlanması bu doküman kapsamında yapılmıştır.

Kapsam Dokümanı. Temelde yazılım ürün hattının hangi ürünleri içereceğini belirler. TADES kapsamında Teknik Ateş Destek Yazılımları olarak geliştirilmiş ve geliştirilecek olan ürünler müşteri bakış açısıyla ve teknik benzerlikleri açısından QFD Analizi yöntemi kullanılarak değerlendirilmiş ve kapsam belirlenmiştir.

Ölçüm ve Takip Planı.Çalışma sırasında toplanacak veriler ve alınacak ölçümleri içerir.

Risk Yönetim Planı.İdari ve Teknik Risklerin nasıl yönetileceği bilgisini içerir.

Teknoloji Öngörüsü Dokümanı.Özellikle Ağ Merkezli Harp ve Servis Yönelimli Mimari konularının yapılacak olan çalışmaya etkileri bu dokümanda incelenmiştir. **Süreçlere Uyum Planı.**Aselsan'da uygulanan mevcut yazılım ve sistem geliştirme süreçlerinin yeni yaklaşımla kullanılıp kullanılamayacağı değerlendirilmiş ve mevcut süreçlere nasıl uyulabileceği bu dokümanda adreslenmeye çalışılmıştır.

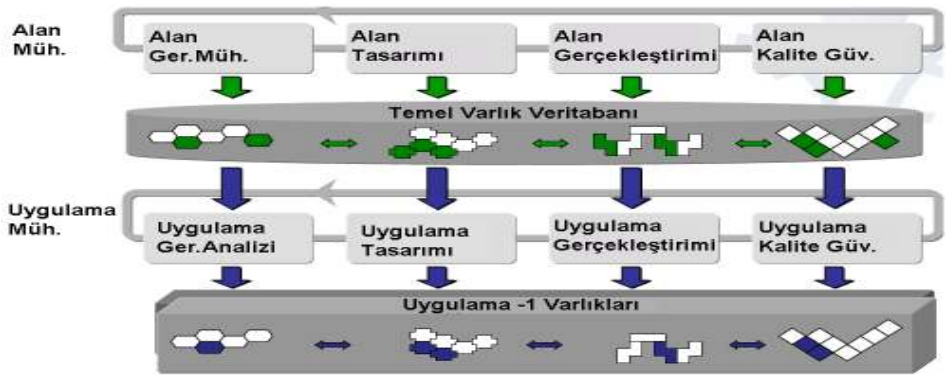
Sponsorluk. Daha önceki bölümlerde TADES kapsamında yazılım ürün hattına geçişin bir ürünü geliştirirken olacağı kararına varıldığı belirtilmiştir. Çalışmaları finanse edecek iki adet proje belirlenmiş ve çalışmaların bu projeler kapsamında yürütülmesine karar verilmiştir.

Araçlar Listesi. Bu çalışma kapsamında TADES'te yazılım geliştirme sırasında kullanılacak araçlar listelenmiş, ayrıca yazılım ürün hattı yaklaşımını destekleyecek hazır araçların araştırılması işleri yapılmıştır. Böylece TADES kapsamında kullanılacak tüm araçların listesi oluşturulmuştur.

Organizasyonel Yapılanma. Bu çalışma kapsamında Yazılım Ürün Hattı organizasyonunda yer alacak yeni roller ve bu rollerin yerine getireceği sorumluluklar belirlenmeye çalışılmıştır.

Uygulama Safhası

Yazılım ürün hattı mühendisliği süreci birbirine paralel işleyen iki süreç şeklinde ifade edilmektedir.[2] Bu iki süreçten birincisi *Alan Mühendisliği* olarak adlandırılmakta ve ortak kullanılacak altyapıyı oluşturmaya yönelik işletilmektedir. İkinci süreç ise *Uygulama Mühendisliği* adını almakta ve ürün geliştirmede kullanılacak ortak varlıklar kümesinin önceden belirlenmiş mekanizmalarla bir araya getirilmesi ve bir takım genişlemeler/konfigürasyonlar yoluyla nihayi ürünlerin oluşturulması için işletilmektedir. Yazılım Ürün Hattı için tanımlanan bu paralel iki süreç aşağıdaki şekilde gösterilmektedir;

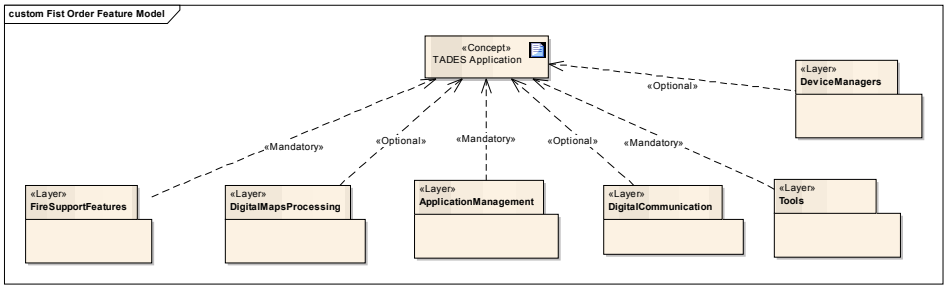


Şekil 1. Yazılım Ürün Hattı Süreci[2]

Uygulama safhasına geçildiğinde yukarıda da belirtilen süreç uyarınca alan mühendisliği çalışmalarına başlanmıştır. TADES çalışmasında daha önceden de

belirtildiği üzere alan mühendisliği ve uygulama mühendisliği çalışmalarının paralel olarak yürütülmesine karar verilmişti. Bu nedenle daha önceden planlama safhasında belirlenen kapsam analizi içerisinde yer alan iki ürün temel alınarak çalışmalara başlanmıştır.

İlk etapta Alan Gereksinim Analizi çalışması yürütülerek kapsama alınan ürünler çerçevesinde bir ürün özellik ağacı oluşturulmaya çalışılmıştır. Burada Teknik Ateş Desteği Alanının geniş bir alan olması ve kapsama alınan ürün sayısının da çok olması nedeniyle ürün ağacı oluşturulması çalışmasının iki aşamalı yapılmasına karar verilmiştir. İlk aşamada oluşturulacak ürün ağacı 1. seviye olarak nitelendirilecek ve Yazılım Konfigürasyon Birimi(YKB) seviyesinde yetenekleri belirlemeye yönelik detayda olacaktır. Bu birinci seviye yetenek ağacının YKB'leri belirlemesi nedeniyle de Referans Mimari çalışmasına da girdi olacağı değerlendirilmiştir. İkinci aşamada ise Referans Mimari'nin de belirlenmesiyle YKB detaylarına girilmesi ve YKB özelinde daha detaylı bir alan analizi ile yetenek ağacının detaylandırılması hedeflenmiştir. Aşağıda 1. seviye yetenek ağacından bir örnek gösterilmektedir.

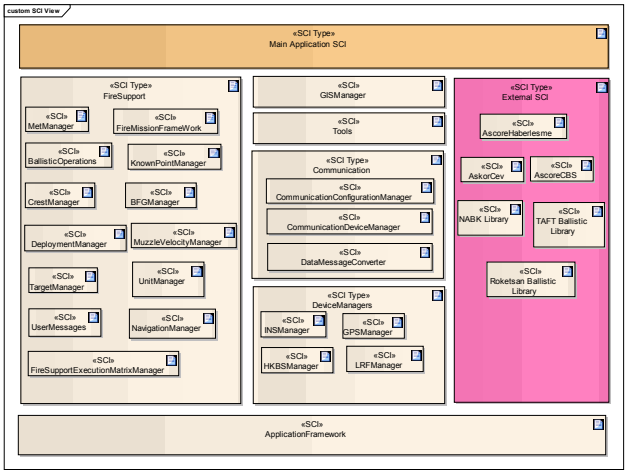


Şekil 2. TADES 1.Seviye Yetenek Ağacı

Alan mühendisliğinin ikinci adımı olan Alan Mimari Tasarımı adımı 1.seviye yetenek ağacının oluşturulmasından sonra başlamıştır. Bu aşamada da Referans Mimari Tasarımın oluşturulması için daha önceden geliştirilen bir teknik ateş desteği yazılım projesinde de kullanılarak tecrübe edinilen Siemens 4 Görünüm Mimari Tasarım Yaklaşımı[4] kullanılmasına karar verilmiştir. Bu yaklaşım ve daha önce kullanıldığı projede elde edilen geçmiş tecrübeler UYMK 08'de [5] ile sunulmuştur. Referans Mimarinin oluşturulması için Siemens 4 Görünüm yaklaşımındaki etkenler stratejiler analizi tüm ürünler düşünülerek yapılmış ve böylece tüm ürünlerin ortak mimarisini oluşturacak Referans Mimari oluşturulmuştur.

Bu çalışma sırasında yaklaşım olarak Siemens 4 Görünüm yaklaşımı temel alınsa da ürün hattı yaklaşımının doğası gereği bu yaklaşımda geçen görünümlerden hangilerinin kullanılacağı ve hangi yeni görünümlerin eklenmesi gerektiği değerlendirilmiştir. Paydaş sayısının artışı ve diğer paydaşlara da hitap etmek düşüncesiyle kullanılacak görünümler şu şekilde belirlenmiştir; *Özellik Modeli Görünümü, Kavramsal Tasarım Görünümü, Parçasal (Modül) Tasarım Görünümü, Kodlama Tasarımı Görünümü, Yazılım Konfigürasyon Birimi(YKB) Görünümü*

Aşağıda referans mimariyi ifade etmek üzere oluşturulan görünümlerden YKB görünümüne ait şekil örnek olarak yer almaktadır.



Şekil 3. TADES Referans Mimarisi YKB Görünümü

Alan tasarımı aşaması da geçildikten sonra belirlenen Referans Mimari de kullanılarak alan gerçekleştirimi aşamasına geçilmiştir. Bu aşamda YKB bazında görev atamaları yapılarak her YKB özelinde öncelikle eksik kalan detaylı alan analizi çalışması yapılması ve yetenek ağacının detaylandırılması ile işe başlanması hedeflenmiştir. Yetenek ağacının alt dalları analiz sonucunda oluştukça her bir yeteneğin nasıl kodlanacağı, nasıl yapısal ve davranışsal özellikler göstereceği Referans Mimari'de belirtildiği şekilde geliştirilmeye başlanmıştır. Geliştirme çalışmaları sonrasında da çoğunlukla doğrulamalar için altyükleniciler kullanılarak YKB testleri yaptırılmıştır.

5.2 Temel Varlıkların İdamesi ve Ürün Geliştirme

Bu adımda ürün hattının altyapısını oluşturan temel varlıkların nasıl idame edileceği ve ürün geliştirme işleminin bu yaklaşım ile nasıl değişeceğinin belirlenmesi ve kayıt altına alınmasına yönelik çalışmalar yürütülmüştür. Temel varlık veritabanı oluşturma yaklaşımımızın aslında ilk ürünü geliştirirken oluşturulacağı daha önceden belirtilmişti. Dolayısıyla temel varlıkların oluşturulması ile ilk ürünün geliştirilmesi de bir anlamda bu adımda gerçekleştirilmiştir. Bu nedenle ilk ürünü oluştururken biraz daha geniş kapsamlı bir analiz yapılmış ve ilk ürün dışında ürün hattı kapsamına giren diğer ürünlerin ihtiyaçlarını da içeren bir mimari tasarım ve gereksinim analizi yapılmıştır. Daha sonra bu çalışmaların sonunda ilk ürünün ihtiyaçlarını karşılayacak şekilde referans mimari tasarım özelleştirilerek ürün mimarisi oluşturulmuş ve yeniden kullanım varlıkları ilk ürünün ihtiyaçları çerçevesinde önceliklendirilerek temin edilmeye(yap/satın al/ altyükleniciye ver/hazır al) başlanmıştır. İlk ürünün geliştirilmesinden sonra her yeni ürünün geliştirilmesi ortak kullanım varlıklarının ürünlerin ihtiyaçları çerçevesinde sayısının arttırılarak geliştirilmesi anlamına gelecektir. İlk aşamada tanımlanan mimari ve bu mimari sonucunda belirlenen tüm yeniden kullanım varlıkları oluşturulduğunda ilk aşamanın da tamamlanmış olacağı varsayılmıştır.

Bu aşamadan sonra yeni bir ürünün TADES ürün hattından üretilmesine yönelik uygulanacak prosedürler, yaklaşımlar ve prensipler bu çalışma süresince belirlenmeye çalışılmıştır. Bu aşamada ürün hattı varlıklarının idamesi ve ürünlerin geliştirilmesi ile ilgili olarak belirlenen prensiplerin yeni ürünlerin geliştirilmesi sırasında geçerli olacağı değerlendirilmektedir. Bu prensiplerin yaygınlaşması ve düzgün bir şekilde uygulanabilmesi için çeşitli destekleyici dokümanlar bu çalışma kapsamında oluşturulmaya çalışılmıştır. Bu kapsamda *TADES Aday Kullanıcıları Değerlendirme Süreci, TADES Kodlama Standardı, TADES Grafiksel Kullanıcı Arayüzü Standardı, TADES Geliştirici El Kitabı* gibi dokümanlar hazırlanmıştır.

5.3 Ürün Hattının İdamesi ve Geliştirilmesi

TADES çalışmasında, ürün hattının temel varlıklarını geliştirme aşaması, kapsamdaki tüm ürünlerin üretilmesi ile gerçekleştirilecektir. Bu adımın tamamlanması ürün hattının tamamlanması ve işletmeye alınması anlamına gelecektir. Bu aşamadan sonra yeni ürünlerin bu ürün hattı kapsamından çıkarılması normal bir süreç olarak devam edecektir. Ürün Hattından yeni ürünler çıktıkça, edinilen tecrübeler ile ürün hattını destekleyici, yeniden kullanılabilir varlıklar ortaya koymak da mümkün olacaktır. Bunlar arasında yazılım konfigürasyon yönetimini kolaylaştıracak yazılımlar, son ürünlerin kullanım için hazırlanmasında konfigürasyon ayarlarını otomatik olarak yapmaya yönelik yazılımlar olabileceği gibi Teknik Ateş Destek Alan Bilgisinin dokümante edilmesine yönelik(sözlük, ontoloji) dokümanların da olabileceği değerlendirilmektedir.

Her yeni ürünün ürün hattı kullanılarak geliştirilmesi, ürün hattının yukarıda bahsi geçen ürünlerinin iyileştirilmesi ve geliştirilmesi anlamına gelecektir. Her yeni ürünün geliştirilmesinde ürün hattı varlıklarının kullanılması ile kullanıcılardan geliştirmeye ve iyileştirmeye yönelik önerilerin toplanması ve özellikle ürün hattı geliştirici grubu tarafından değerlendirilerek uygulamaya alınması gerçekleştirilecektir. Ayrıca temel varlıkların her kullanımında kullanıcıların görüşleri çerçevesinde kullanımını kolaylaştırmaya yönelik dokümantasyon ve eğitim gibi mekanizmaların geliştirilmesine yönelik çalışmaların da yapılacağı değerlendirilmektedir.

6 Yazılım Ürün Hattına Geçiş Aşamasında Edinilen Tecrübeler

TADES Yazılım Ürün Hattı çalışmasının başlangıcından bugüne kadar her aşamasında çok çeşitli tecrübeler edinilmiş ve gözlemlerde bulunulmuştur. Yazılım Ürün Hattı oluşturma çabasında en çok altyapıyı oluşturma aşamasının sancılı geçmesi ve bu aşamadan dersler çıkarılabileceğini tahmin etmek pek de zor olmayacaktır. Biz TADES çalışmasının bu aşaması içerisinde olarak aslında ağırlıklı olarak yaşanan sıkıntılardan bahsedeceğiz.

Öncelikle çalışmanın başlangıç aşamasında, yazılım ürün hattı kavramı konusunda tüm ekibin aynı bilgi seviyesinde olmayışı çalışmaların yavaş ilerlemesine, belirli noktalarda zaman kaybına yol açan sonuçsuz tartışmalara neden olmuştur. Yazılım ürün hattı yaklaşımını uygulama kararı alınmadan önce bu konuda çalışacak olan ekipten iki kişi, biri daha detaylı olmak üzere bu konuda eğitim almıştır. Çalışma başlatıldıktan sonra hazırlanan bir eğitim planı çerçevesinde eğitim almış kişiler tarafından tüm ekibe yazılım ürün hattı kavramı anlatılmış ve eğitim verilmiştir. Çalışmaya daha sonradan

katılanların olması ve bu kapsamda eğitimin tekrarlanmamış olması zaman zaman yazılım ürün hattı ile ilgili çok temel kavramların bile ekip içinde tartışılır olmasına neden olmuştur.

Çalışma sırasında organizasyonel yapılanma değişikliği ile ilgili çalışmalar yapılsa da firma kültürü ve şartları çerçevesinde değişiklik tam olarak hayata geçirilememiştir. Sadece roller belirlenmiş fakat tüm ekip elemanları hemen her türlü rolde görev almak zorunda kalmıştır. Bu da kaynak kullanımı konusunda sıkıntılar yaratmıştır.

TADES çalışması başlangıcında çalışma sırasında kullanılacak araçlar belirlenmiş ve listelenmiştir. Bu araçlardan özellikle destekleyici nitelikte olan araçların çalışmanın ilerleyen aşamalarında çok da uygun olmadığı değerlendirilerek değişik araç arayışları içerisine girilmek zorunda kalınmıştır. Bu da zaman kaybı yaratmıştır.

ASELSAN A.Ş savunma sanayinde büyük ve köklü bir firma olmasına karşın her firmada olduğu gibi zaman ve işgücü anlamında sıkıntılar yaşayabilmektedir. Altyapıya ve AR-GE çalışmalarına kaynak ayırmakla birlikte, özellikle bu çalışma sırasında alt yapıya harcanan kaynaklar aynı zamanda proje için harcanan kaynakla aynı olmak durumunda kalmıştır. Bu da altyapı çalışmalarının detayı ve özenine doğrudan yansımıştır. Bu nedenle oluşturulan özellik modelinin sık sık değiştirilmesi, referans mimari kapsamında alınan kararların zaman içerisinde değişik problemlerle karşılaştıkça değiştirilmesi gibi sonuçlar doğurmuştur.

ASELSAN salt bir yazılım firması değil sistem geliştiren ve entegrasyonunu yapan bir firmadır. Organizasyonel yapılanması itibariyle de Yazılım Mühendisliği Bölümü, Sistem, Elektronik Tasarım, Elektronik Donanım ve Test Mühendisliği gibi birçok fonksiyonel bölümle birlikte çalışmakta ve her bölüm birbirine bağlı süreçler çerçevesinde işini yürütmektedir. Bu da Yazılım süreçlerinin diğer bölüm süreçlerine bağlı bir şekilde işletilmesini gerektirmektedir. Bu da Yazılım Ürün Hattı oluşturma probleminin yazılım dünyası boyutunu aşan bir hale dönüşmesine neden olmuştur.

TADES yazılım ürün hattı oluşturma çalışması başlarında ürün geliştirme yaklaşımındaki değişikliğin mevcut süreçlere uyum planı çerçevesinde var olan süreçlere ciddi bir değişiklik getirmeyeceği öngörülmüştür. Süreçlerde yapılması gerekebilecek değişikliklerin de ilk birkaç ürünün çıkışından sonra belirlenebileceği öngörülmüştür. Bu öngörü, mevcut süreçlerin yazılım ürün hattı gibi bir yaklaşım düşünülerek hazırlanmamış olması nedeniyle ve işin içine salt yazılım süreçleri değil sistem geliştirme v.b. başka süreçlerin de girmesi sebebiyle özellikle ilk ürünün dokümantasyonunun oluşturulması konusunda ciddi sıkıntılar doğurmuştur.

Yukarıda bahsedilen tecrübeler ve sıkıntılara rağmen çalışma ile elde edilen pek çok kazanım olduğu değerlendirilmektedir. TADES'ten henüz tek bir ürün çıkmış olması nedeniyle ölçülebilir veriler elde edilememiş olsa da aşağıda bu kazanımlar aktarılmıştır.

Teknik Ateş Destek Komuta Kontrol yazılımlarının geçmişte geliştirilen yazılımlara göre daha esnek, değişikliğe açık ve ortak bir mimarisi olmuştur.

Çalışma sırasında ortak kullanım varlıklarının son yazılım ürünlerinin geliştirilmesi sırasında kullanılacak kodlama standardının belirlenmiş olması ve bu standarda uyumun gözetilmesi geçmiş yazılımlara göre kodlama kalitesini arttırmıştır.

Geçmişte farklı kişilerce geliştirilen pek çok farklı yazılım ürününün yeni bir ekiple idame edilmesi problemi ortadan kalkmıştır. Ortaklamanın geçmişte istenen oranda yakalanamamış olması aynı yeteneklerin farklı ürünlerde tekrar yazılması gibi durumları da doğurduğu için bir yerde çıkan bir hatanın benzer diğer ürünlerde de giderilmesi problemini doğurmaktaydı. Bu sıkıntının bu yaklaşımla ortadan kalktığı değerlendirilmektedir.

Her ne kadar geliştirilen altyapıdan daha sadece 1 adet nihai ürün çıkmış olsa da önümüzdeki 2-3 yıllık süreçte, bu sayının 15'e kadar çıkması beklenmektedir. Bu ilerleme sonrasında ortaya çıkan ürünlerin daha kaliteli olması beklenmektedir.

Çalışmanın başlangıcında temel gerekçelerden biri olan kaynak sıkıntısının tam olarak olmasa da bu yaklaşımla bir nebze hafifletilmiş olduğu değerlendirilmektedir. Şu ana kadar yapılan çalışmalar altyapıya yönelik olduğu için ürün hattı oluşturma çalışmasının başlangıcında yaşanan ilave işgücü harcanması söz konusu olmuştur. Bu nedenle henüz net olarak kaynak sıkıntısının azaldığı hissedilememiş olmakla beraber ürün hattından çıkan ürün sayısı arttıkça bu faydanın daha hissedilir olacağı değerlendirilmektedir.

7 Sonuç

Yazılım Ürün Hattı yaklaşımı dünyada oldukça popüler hale gelmiş ve kendini ispatlamış bir yazılım geliştirme yaklaşımıdır. Bu yaklaşımın askeri alanda yazılım geliştiren firmalar için de uygun bir yaklaşım olduğu düşünülmektedir. Bu fikirden hareketle ASELSAN'da Teknik Ateş İdare Komuta Kontrol yazılımları geliştirilmesi üzerine oluşturulmaya çalışılan yazılım ürün hattı süreci ve bu süreçte yaşanan tecrübeler ve edinilen kazanımlar bu makalede aktarılmaya çalışılmıştır. İlerleyen aşamalarda bu çalışma kapsamında edinilen tecrübeler doğrultusunda ASELSAN'da Yazılım ve ilişkili diğer süreçlerde değişiklik önerilerinin oluşturulması hedeflenmektedir.

Kaynakça

1. Paul Clements, Linda Northrop, Software Product Lines Practices and Patterns, Addison Wesley, 2002 ISBN: 0201703327.
2. K. Pohl, G. Böckle, and F. v. d. Linden; Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, Berlin Heidelberg New York, 2005.
3. Sholom Cohen, Guidelines for Developing a Product Line Concept of Operations 1999

   CMU/SEI-99-TR-008
4. C. Hofmeister, R. Nord, and D. Soni, Applied Software Architecture, Boston: Addison-Wesley, 2000 ISBN: 0201325713.
5. E.Barak, S.Erdem, O.Dayıbaş,T.Koray Komuta Kontrol Yazılımlarında Mimari Tasarım Yöntemi Uygulama Deneyimi UYMK 2008

Yazılım Ürün Hattı Değişkenliğinin Denetim Çevrimi ile Ele Alınması

Emra Aşkaroğlu[1] Kemal Memiş[2] Mustafa Özpınar[3]

[1,2,3]ASELSAN A.Ş. Savunma Sistem Teknolojileri Grubu

1e-posta: easkaroglu@aselsan.com.tr 2 e-posta: kmemis@aselsan.com.tr
3 e-posta: mozpinar@aselsan.com.tr

Özet. Günümüz teknoloji dünyasında, birbirine benzer fonksiyonel özelliklere sahip sistemlerin yazılım ürün hattı yapısı içerisinde ele alınarak ürün maliyetini düşürmek ve yeniden kullanılabilirliği (reusability) azami seviyede tutarak sistemleri daha kısa sürede oluşturmak yazılım geliştiricilerin en büyük hedeflerinden birisidir.

Bu bildiride ASELSAN A.Ş. Savunma ve Sistem Teknolojileri (SST) Grubunun komuta kontrol sistemleri için yazılım ürün hattı yaklaşımı kullanılarak geliştirmiş olduğu Teknik Ateş Destek Sistemleri (TADES) projesi kapsamında yapılan çalışmalar ve özellikle bu çalışmalar sırasında yazılım ürün hattında kod seviyesinde değişkenliğin nasıl tanımlandığı ve ele alındığı anlatılacaktır.

1 Giriş

Yazılım ürün hattı, ortak özellikleri fazla olan bir ürün ailesindeki ortaklıkları ve değişkenlikleri belirleyerek, bu ortaklık ve değişkenliklere göre yazılım birimlerinin tanımlanması ve üretilecek ürünlerde bu yazılım birimlerinin kullanılmasıdır [1]. Diğer bir ifadeyle bir yazılım mühendisliği kavramı olan yazılım ürün hattı, ürün ailesindeki ortaklıklara ve değişkenliklere göre yazılım birimleri içeren bir havuz oluşturulması ve yazılım ürün hattından çıkarılacak ürünlerin bu havuzdaki yazılım birimlerini kullanmasıdır. Yeniden kullanım özelliği, yazılım ürün hattının ürün aileleri ile uğraşan organizasyonlara sağladığı en önemli yararlardan biridir.

Yazılım ürün hattı oluştururken, bu ürün hattının kapsamı belirlendikten sonra dikkat edilmesi gereken en önemli konu ürün hattı havuzuna eklenecek yazılım birimlerinin belirlenmesidir. Bu yazılım birimlerinin belirlenmesi için öncelikle yazılım ürün hattından üretilmesi düşünülen ürün ailesinin ortaklıklarının belirlenmesi, değişkenliklerinin tanımlanması ve ele alınması gerekir.

Yazılım ürün hattı yaklaşımı düşünüldüğünde, bileşenlerin birbirinden bağımsız gerçeklenebilmesi ve çalışabilir olması için bağımlılık işlemlerini ele alacak tasarımlara ya da araçlara ihtiyaç duyulur. Spring Çatısı bu ihtiyaçları karşılayacak gelişmiş ürünlerden birisidir.

Spring, nesneye dayalı programlama dillerinde (C#, Java vb.) nesneleri XML tabanlı bir yapı kullanarak bağlamaya yarayan bir araçtır[2]. Spring, yaklaşım olarak Denetim Çevrimini (Inversion of Control) ve Bağımlılık İletimi (Dependency Injection) tasarım örüntüsünü kullanır. Denetim Çevriminde temel amaç, bir bileşenin diğer bileşenlere bağımlılığını dışarıdan müdahaleyle sağlamaktır. Bağımlı olunan bileşene ait bir nesne Spring tarafından yaratılarak bir arayüz aracılığıyla ihtiyacı olan bileşenlere sağlanır.

Temelde bir bileşen, başka bir bileşenin herhangi bir özelliğini kullanmak istediğinde, o bileşene ait bir nesne yaratarak istenen özelliğe ulaşır. Bu durumda bileşenler birbirlerine tamamen bağımlı hale gelirler. Bağımlılık İletimi örüntüsü ile bu bağımlılık bileşen bazında kaldırılmış olur. Spring, XML yapılandırması kullanarak nesneleri yaratıp referansları bileşenlere dağıttığı için bu bağımlılığı bileşen seviyesinden ürün seviyesine çeker. Her bileşen, Spring'in sağladığı referansları kullanarak diğer bileşenlerin arayüzlerini kullanabilir hale gelir. Bileşenler, hangi bileşene bağımlı olduğunu bilmeden sadece kendisine sağlanan referans nesnenin sağladığı arayüzü bilerek işlem yapar. Bu sayede aynı arayüzü gerçekleyen yeni bir bileşen yazılarak Spring Çatısı'nın XML yapılandırmasında bağımlılık yeni yazılan bileşen üzerinden yapılabilir. Aynı arayüzü sağlayan farklı bileşenler yazılarak yazılım ürün hattında çeşitlilik arttırılabilir. Ayrıca Spring Çatısı nesne atamalarını sağlayacağı için her ürün için yeniden derleme ihtiyacı olmayacaktır.

Bildirinin geneline baktığımızda 2. bölümde TADES Yazılım Ürün Hattı'nda ortaklıkların ve değişkenliklerin nasıl ele alındığı anlatılmaktadır. Bu değişkenliklerin "Denetim Çevrimi" ile nasıl ele alındığı ve Spring Çatısı'nın kullanım örneği 3. bölümde anlatılmaktadır. Doküman Sonuç bölümüyle sonlanmaktadır.

2 Yazılım Ürün Hattı Değişkenliği

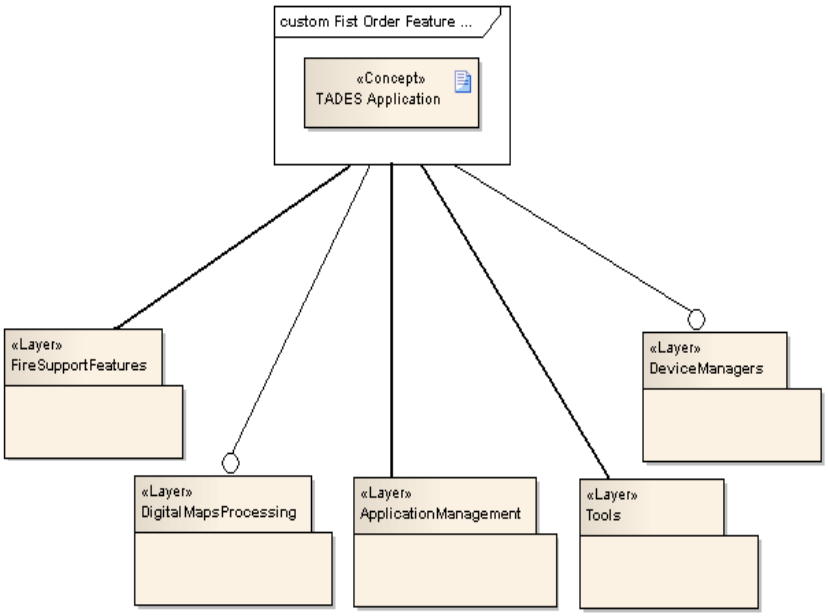
Yazılım ürün hattı değişkenliği ele alınırken öncelikle yazılım birimi özelliklerinin belirlenmesi ve bu özelliklerin ürünlerdeki ortaklıklara ve değişkenliklere göre kategorize edilip ürün özellik ağacının çıkarılması gerekmektedir. Özellikler belirlenirken Tablo 1'deki özellik tipleri kullanılabilir[3]. Özellik ağacının oluşturulması, kodlama aşamasında kullanılacak tekniklerin seçimi konusunda stratejik kararlar verilmesine neden olacağı için oldukça önemlidir. Tablodaki birbirini kapsayan (mutually inclusive) ve birbirini dışlayan (mutually exclusive) tipleri diğer 3 özellik tiplerinden ayrılmalıdır. Çünkü zorunlu (mandatory), seçenekli (optional) ve alternatif (alternative) tipleri bir özelliğin durumunu diğer özelliklerden bağımsız bir şekilde belirtirken, mutually inclusive ve mutually exclusive tipleri bir özelliğin durumunu diğer özelliklere göre belirler.

Tablo 1: Özellik Tipleri

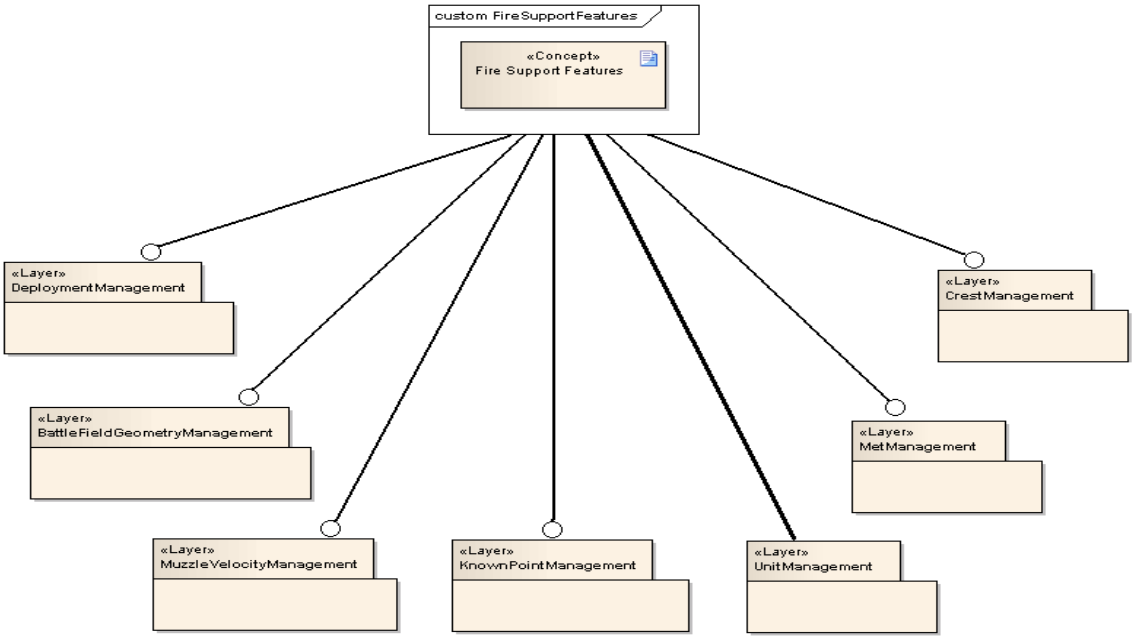
Özellik Tipi	Anlamı	Kullanılan Şekil
Zorunlu (Mandatory)	Özellik mutlaka olmalıdır.	
Seçenekli (Optional)	Özelliğin kullanımı tün sistemden bağımsızdır. Kullanımı isteğe bağlıdır.	
Alternatif (Alternative)	Özellik eklendiğinde diğer özellik yerine kullanılabilir	
Birbirini Kapsayan (Mutually Inclusive)	Özelliğin kullanılabilmesi için başka özelliklerin de kullanılıyor olması gerekmektedir.	
Birbirini Dışlayan (Mutually Exclusive)	Özelliğin kullanılabilmesi için başka özelliklerin kullanılmaması gerekmektedir.	

TADES yazılım ürün hattından üretilmesi öngörülen ürünler göz önüne alındığında, tüm ürünlerin alt yapısını oluşturacak "ApplicationManagement" özelliği zorunlu olarak belirlenmiştir. Aynı şekilde, her üründe hesaplama araçlarının olacağı düşünüldüğünde "Tools" özelliğinin de zorunlu olması sonucu doğmuştur. Şekil 1'de ürün özellik ağacının birinci seviyesi gösterilmektedir.

Şekil 2'de, Şekil 1'de verilen "FireSupportFeature" özelliğinin alt özellikleri verilmiştir. Şekil 2'de görüleceği gibi birim tanımlama ve birimlerle ilgili işlemleri ele alan "UnitManagement" özelliği zorunlu olarak belirlenmiştir. Fakat ilk hız yönetimini ele alan ve ilk hız verileriyle yapılacak işlemleri ele alan "MuzzleVelocityManagement" özelliği üretilecek tüm ürünlerde olmayacağı için TADES Yazılım Ürün Hattı'nda seçenekli bir özellik olarak belirlenmiştir.



Şekil 1: Ürün Özellik Ağacı 1. Seviye



Şekil 2: Ürün Özellik Ağacı 2. Seviye

TADES projesinde ürün hattı havuzu, bileşenler yerine Yazılım Konfigürasyon Birimlerinden (YKB) oluşmaktadır. Yazılım Konfigürasyon Birimleri aynı amaç için kullanılan bileşenlerin oluşturduğu yapılardır. MVC (Model-View-Controller) tasarım örüntüsünün kullanıldığı TADES projesinde her özellik için Model, View ve Controller katmanları oluşturulmuştur. Bu 3 katman ayrı bileşenler olsalar da TADES yazılım ürün hattı için bu 3 katmanın birleşmesiyle YKB oluşmaktadır.

Yazılım ürün hattı ortaklıkları ve değişkenlikleri belirlendikten ve ürün özellik ağacı çıkarıldıktan sonra, kodlama aşamasına geçilmeden önce bu özellikleri gerçekleştiren bileşenlerin birbirlerine bağlanma zamanlarının (binding time) belirlenmesi gerekmektedir. Bu belirleme kodlama aşamasında bileşenler arası iletişimin nasıl ve ne zaman olacağını etkileyecektir.

Bağlanma zamanları aşağıdaki gibi kategorize edilebilir[3]:

- Derleme zamanı (Compile-time)
- Bağlama zamanı (Link-time)
- Çalışma zamanı (Runtime)
- Çalışma zamanı sonrası-güncelleme zamanı (Post-runtime/Update-time)

TADES düşünüldüğünde her YKB diğer YKB'ler ile iletişim sağlamak için arayüz sınıfları içerecektir. YKB'ler bu arayüzleri kullanarak diğer YKB'lerden istedikleri özellikleri kullanabilecektir. Denetim çevrimi örüntüsü sayesinde referanslar dışarıdan sağlanacağı için YKB'ler diğer YKB'lere ait nesneleri kendileri yaratmayacak ve bu YKB'lere bağımlı olmayacaklardır. TADES kapsamında yukarıdaki bağlanma zamanı seçeneklerinden "Çalışma Zamanı" uygun görülmüştür. Bu durumda her YKB'ye ait nesneler çalışma zamanında yaratılıp, bu nesnelerin referansları, ilgili özelliği kullanacak diğer YKB'lere verilecektir. Fakat çıkarılacak ürünlerde bir yeteneğin seçeneğe bağlı olup olmaması durumu göz önünde bulundurularak tüm YKB'lerde referans olarak tutulan diğer YKB nesnelerinin yaratılıp yaratılmadığı kontrol edilecektir. Bu durumda eğer nesne yaratılmamışsa, ilgili yeteneğin üründe olmadığı anlamına gelecektir. Örneğin elimizde a özelliğini sağlayan A YKB'si ve b özelliğini B YKB'si olsun. B YKB'sinin a özelliğini de kullanacağını düşünelim. Denetim Çevrimi örüntüsü ile A ve B YKB'lerinden nesneler yaratılarak A YKB'sinden yaratılan nesnenin referansı B YKB'sine verilerek a özelliğinin kullanımı sağlanır. Fakat eğer çıkarılacak üründe a özelliğinin olmaması durumu olursa A YKB'sinden bir nesne yaratımı söz konusu olmayacaktır. Bu durumda B YKB'si A nesnesinden alacağı referansın yaratılıp yaratılmadığını kontrol ederek a özelliğini kullanmadan işlemlerini yapmaya devam edebilecektir. Bu örnek, bağımlılığın Denetim Çevrimi ile ele alınmasının en önemli yararını ifade etmektedir.

Çalışma zamanında nesnelerin yaratılması ve bu nesnelerin YKB'lere referans olarak verilmesi işlemlerinin detayları 3. bölümde anlatılmıştır.

3 Değişkenliğin Denetim Çevrimi ile Ele Alınması

Günümüz yazılım geliştirme çalışmaları açısından baktığımızda, büyük ve geniş çaplı projeler, daha küçük parçalara ayrılıp farklı geliştiriciler tarafından kimi zaman farklı geliştirme ortamları kullanılarak geliştirilmektedir. Proje parçalarının farklı yazılımcılar tarafından geliştirilmesi, farklı geliştirme ortamlarının kullanılması, her parça için farklı yazılım kurallarının kullanılması bu parçaların birleştirilmesini zorlaştırmaktadır. Örneğin TADES projesi için çıkarılan ürün özellik ağacındaki özellikleri gerçekleştiren YKB'ler farklı yazılımcılar tarafından geliştirilmiştir. Geliştirme esnasında bir yazılımcının diğer YKB'lerin gerçeklenmesini beklemeden ya da diğer YKB'lerin geliştirme ortamından ve yönteminden bağımsız bir şekilde kodlama yapabilmesi en önemli sorunlardan birisiydi.

Denetim Çevrimi, proje parçaları ya da uygulamalar arasındaki iş akışının ve bağımlılığın yönetilmesini sağlar[4][5]. Denetim çevriminin sağladığı en önemli özellik, ihtiyaç duyulan nesnenin kullanılacak nesne tarafından bilinmemesine olanak sağlamasıdır.

Denetim Çevrimi, birçok yöntemle yapılabilmektedir [4][5].

- Arayüz İletimi (Interface Injection): Bu yöntemde bağımlılıkların iletimi için arayüzler tanımlanır. Hizmetler ve bileşenler için arayüzler tanımlanıp bu arayüzler ilgili bağımlı olunan sınıflarca gerçekleştirilir.
- Kurucu İletimi (Setter Injection): Bu yöntemde yapılan şey, yararlanılacak hizmet ve ya kullanılacak bileşen için setter metodlarının yazılması ve yaratımlar için yapılandırma dosyalarının kullanılmasıdır.
- Yapıcı İletimi (Constructor Injection): Bu yöntemde sınıf yapılandırıcıları tüm bağımlılıkları içerir. Bağımlı olunan sınıflara ait nesneler bu yapılandırıcının parametreleri olarak kullanılırlar.

SPRING Çatısı Denetim Çevriminde "Kurucu İletimi" ve "Yapıcı İletimi" yöntemlerini desteklemektedir. TADES projesinde SPRING Çatısı yardımıyla Kurucu İletimi yöntemi kullanılmaktadır. Tüm bileşenler bağımlı oldukları bileşenler için Setter metodları tanımlayarak spring tarafından yaratılan nesnenin referansını elde etmektedirler.

TADES Yazılım Ürün Hattı'nda üretilen tüm ürünlerde, bileşen bağımlılıklarını ele alan bir yapılandırma dosyası (applicationcontext.xml) bulunmaktadır. Bu XML yapısındaki yapılandırma dosyası SPRING tarafından kullanılarak bağımlılıklar atanmaktadır.

Aşağıda TADES projesinde SPRING kullanımına bir örnek verilmiştir.

```
<object                              id="objBMDeploymentManager"
type="Tr.Com.Aselsan.TADES.FireSupport.DeploymentManager.BMDep
loymentManager,DeploymentBusinessManager">
…
</object>
```

Yukarıdaki yapılandırma dosyası parçası ile Tr.Com.Aselsan.TADES.FireSupport.DeploymentManager.BMDeploymentManager isim uzayı (namespace) içerisinde tanımlı DeploymentBusinessManager sınıfına ait bir nesne yaratılmaktadır. Bu nesneye erişim ise "objBMDeploymentManager" referansı ile sağlanacaktır.

```
<object id="objBMCoordinateManager"
type="Tr.Com.Aselsan.TADES.ApplicationFramework.CoordinateOpera
tions.BMCoordinateManager,  AppFWBusinessManagers">
 </object>
```

Benzer şekilde yukarıdaki yapılandırma dosyası parçası ile Tr.Com.Aselsan.TADES.ApplicationFramework.CoordinateOperations isim uzayı (namespace) içerisinde tanımlı BMCoordinateManager sınıfına ait bir nesne yaratılmaktadır.

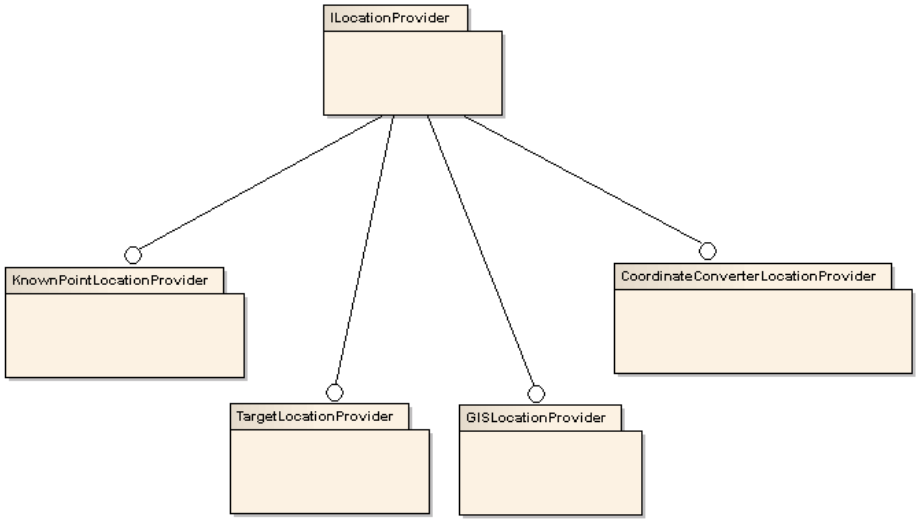
```
<object id="objBMNavigationManager"
type="Tr.Com.Aselsan.TADES.FireSupport.NavigationManager.BMNavi
gationManager, NavigationBusinessManager">
       <property name="BMCoordinateManager"
ref="objBMCoordinateManager" />
       <property name="BMDeploymentManager"
       ref="objBMDeploymentManager" />
</object>
```

Yukarıdaki bölümde de bağımlılık iletimi örneği görülmektedir. Örnekte Tr.Com.Aselsan.TADES.FireSupport.NavigationManager isim uzayı içerisinde tanımlı BMNavigationManager sınıfı objBMDeploymentManager ve objBMCoordinateManager referanslarıyla tanımlı nesnelere bağımlıdır. Bu yapılandırma dosyası ile yaratılan objBMDeploymentManager ve objBMCoordinateManager nesne referansları BMNavigationManager sınıfı içerisindeki BMCoordinateManager ve BMDeploymentManager ile tanımlanan Setter metodlarına enjekte edilir. Böylece BMNavigationManager sınıfı içerisinde nesne yaratım işlemi ele alınmaz. SPRING Çatısı çalışma zamanında (run-time) nesne yaratım işlemlerini gerçekleştirerek bağımlılıkları tanımlar.

Bağımlılık iletimi ile sadece nesne atamaları değil liste atamaları da gerçekleştirilebilir. Aşağıdaki verilen örnekte konum sağlama özelliği içeren birden çok bileşeni liste şeklinde içermesi gereken bağımlı bir bileşen için bağımlılık iletiminin ele alınması gösterilmiştir.

```
<property name ="LocationProviderList">
      <list element-
type="Tr.Com.Aselsan.TADES.Interfaces.ILocationProvider,
AppFWBMInterfaces">
             <ref local="objGMTargetManager"/>
       <ref local="objGMKnownPointManager"/>
      </list>
</property>
```

Yapılandırma dosyasında bir "object" tanımı içerisine yukarıdaki tanımlamanın yazılması ile ILocationProvider türüne ait LocationProviderList adında bir listeye bağımlılık iletimi gerçekleştirilir. Burada önemli olan şey objGMTargetManager ve objGMKnownPointManager refernslarıyla tanımlanmış nesnelerin ILocationProvider arayüzünü gerçekleştirmiş olmasıdır. Böylece LocationProviderList listesindeki tüm elemanlar için, elemanların referanslarının hangi nesneden geldiği bilinmeksizin aynı özellik kullanılabilecektir. ILocationProvider arayüzünde GetLocation() fonksiyonunun arayüz tanımı olacaktır. Bu arayüz LocationProviderList listesindeki tüm bileşenler için gerçeklenecektir. Bu liste elemanları için kullanılan elemanın nesne tipi ne olursa olsun GetLocation() fonksiyonu kullanılabileceği için bağımlılık bu özellik için tamamen kalkacaktır.



* Şekil 3: LocationProvider özellik ağacı

Şekil 3'te LocationProvider özelliğinin özellik ağacı gösterilmiştir. Ürünlerde hepsi seçeneğe bağlı(optional) olan 4 ayrı LocationProvider bulunabilir. LocationProviderList ile belirttiğimiz yapılandırma dosyası parçası örneğinde bu seçeneklerden 2 tanesi bulunmaktadır. Diğer seçenekler de isteğe bağlı olarak listeye eklenebilir veya listeden çıkarılabilir.

Denetim Çevrimi ile değişkenliğin verilen örnekler doğrultusunda ele alınması TADES Yazılım Ürün Hattı'ndan üretilen ürünler için büyük kolaylıklar sağladığı görülmüştür. Son durumda ürünler oluşturulurken istenen özellikleri gerçekleştiren bileşenlerin alınması ve Spring yapılandırma dosyası ile Bağımlılık İletiminin sağlanması yeterli olacaktır.

4 Sonuç

Bu bildiri ile Yazılım Ürün Hattı'nda değişkenliği nasıl ele aldığımız ve bu değişkenliği ele alırken Spring Çatısı'nın hangi özelliklerini kullandığımız ile ilgili detaylı açıklamalara yer verilmeye çalışılmıştır. Yazılım Ürün Hattı'nda öncelikle ortaklıkların ve değişkenliklerin belirlenmesinin önemi vurgulanmış, bu ortaklıklar ve değişkenlikler belirlenirken gösterimin nasıl olacağına dair örnekler verilmiştir. Ayrıca değişkenliği ele alırken kullanılan Denetim Çevrimi'nin özellikleri ve Denetim Çevrimi'ni pratik bir şekilde yapabilmemizi sağlayan Spring Çatısı aracına dair bilgiler verilmeye çalışılmıştır. TADES projesi kapsamında, geliştirilen Yazılım Ürün Hattı bileşenlerinin değişik kombinasyonlarının kullanılmasıyla birçok ürün oluşturulup çalışma performansları değerlendirilmiştir. Oluşturulan ürünlerin tekrar derlenmeye ihtiyaç olmadan çalıştığı ve bağımlılıkla ilgili bir sorun yaratmadığı görülmüştür.

Kaynakça

[1] Paul Clements,Linda Northrop, Software Product Lines:practices and patterns, 6. Baskı, Nisan 2007, ISBN 0-201-70332-7

[2] www.springsource.org

[3] Márcio de M. Ribeiro , Pedro Matos, Jr. , Paulo Borba, A decision model for implementing product lines variabilities, Proceedings of the 2008 ACM symposium on Applied computing, Mart 16-20, 2008, Fortaleza, Ceara, Brazil

[4] http://en.wikipedia.org/wiki/Inversion_of_control

[5] http://www.martinfowler.com/articles/injection.html

YAZARLAR DİZİNİ