T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM GELİŞTİRMEDE ÜRÜN HATTI MÜHENDİSLİĞİ YAKLAŞIMI KULLANIMI

RECEP ATAŞ

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN
PROF. DR. OYA KALIPSIZ

İSTANBUL, 2011

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM GELİŞTİRMEDE ÜRÜN HATTI MÜHENDİSLİĞİ YAKLAŞIMI KULLANIMI

Recep ATAŞ tarafından hazırlanan tez çalışması ../../…. tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi _____

Prof. Dr. Nizamettin AYDIN

Yıldız Teknik Üniversitesi _____

Doç. Dr. Zuhal TANRIKULU

Boğaziçi Üniversitesi _____

ÖNSÖZ

Üniversitede yapılan akademik çalışmalar ile yazılım sektörü tarafından uygulanan yazılım geliştirme yöntemlerinin birlikte ele aldığını düşündüğüm bu tez çalışmasının, ülkemizde bu alanda daha fazla çalışma yapılarak yazılım üretkenliğinin artırılmasına yardımcı olmasını umuyorum. Bu tez çalışması süresince bilgisini ve yardımlarını esirgemeyen tez danışmanım Sayın Prof. Dr. Oya KALIPSIZ'a teşekkürlerimi sunarım. Ayrıca çalışma süresince hep yanımda olan iş arkadaşlarım ve yöneticilerim Sayın Oğuz Keskin, Zafer Şen ve Taha Özket'e ve çok sevgili aileme yardımları için teşekkürü bir borç bilirim.

Ağustos, 2011

Recep ATAŞ

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

LoC	Bir yazılım ürünündeki kaynak kodların içindeki toplam satır sayısı

KISALTMA LİSTESİ

ADK	Alternatif Dağıtım Kanalları
COTS	Commercial-off-the-shell
CBSD	Component Based Software Development (Bileşen Tabanlı Yazılım Geliştirme)
DSL	Domain Specific Language (Alana Özgü Dil)
DSPL	Dynamic Software Product Lines (Dinamik Yazılım Ürün Hatları)
DSSA	Domain Specific Software Architecture (Alana Özgü Yazılım Mimarisi)
ERP	Enterprise Resource Planning
J2EE	Java 2 Platform Enterprise Edition
OOP	Object-Oriented Programming (Nesneye Dayalı Programlama)
SEI	Software Engineering Institute
SPL	Software Product Line
SPLE	Software Product Line Engineering
WSDL	Web Service Description Language
XML	Extensible Markup Language
VTYS	Veri Tabanı Yönetim Sistemi
YÜH	Yazılım Ürün Hattı
YÜHM	Yazılım Ürün Hattı Mühendisliği

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

YAZILIM GELİŞTİRMEDE ÜRÜN HATTI MÜHENDİSLİĞİ YAKLAŞIMI KULLANIMI

Recep ATAŞ

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Prof. Dr. Oya KALIPSIZ

Son yıllarda yazılımların metalaşması sonucu bir yazılım sistemini oluşturan parçaların büyük bölümü sadece o yazılım tarafından değil, genel birçok yazılım ürünü tarafından da kullanılmaktadır. Büyük bölümü ortak olan yazılım sistemlerinde daha efektif yazılım geliştirme için sadece ürüne özgü kısımlara odaklanılmalıdır. Bu yazılım sistemleri için yazılımın yeniden kullanımının sistematik olarak ele alınması gerekmektedir. Yazılım geliştirme süreçlerini hızlı, düşük maliyetli ve kaliteli bir şekilde gerçekleştirebilmek için yazılımın yeniden kullanımı alanında birçok yeni fikir tanımlanmıştır.

Yazılım mühendisliğindeki tüm güncel yaklaşımlar, yazılım yeniden kullanımının artırılmasını hedeflemektedir. Bu amaçla ortaya çıkan en yeni yaklaşımlardan biri Yazılım Ürün Hattı Mühendisliği yaklaşımıdır. Bu yaklaşım yazılım geliştirme sürecinde önemli bir yer almaya ve organizasyonlar tarafından benimsenmeye başlanmıştır. Yazılım Ürün Hatları, belirli ortak kabulleri sistematik olarak yeniden kullanarak ve ürünlerin farklılık gösterdiği yerlerde değişkenliği yöneterek benzer ürünler geliştirmeyi amaçlamaktadırlar.

Çalışmada ele alınan kurumsal ADK uygulamaları alanı, farklı dağıtım kanallarında çalışabilen çok sayıda benzer özellikleri olan yazılım sistemlerini içerir. Buradan yola çıkarak, bu çalışmada J2EE ve web tabanlı uygulamalarında kullanılan bir yazılım ürün hattı mimarisi incelenerek örnek uygulamalarla bu yöntemin yazılım geliştirmedeki başarısının gösterilmesi amaçlanmıştır. Çalışmada kullanılan Dinamik Yazılım Ürün

Hattı, Servis Tabanlı Mimarinin kullanımıyla değişkenliklerin çalışma anında dinamik olarak yapılandırılması sağlamaktadır. Bu yaklaşımın benzer özellikteki kurumsal ADK yazılımları için yazılım üretkenliğini arttırdığı görülmüştür. Sonuç olarak gereksinim analizi, tasarım, geliştirme ve test süreçlerinde yazılım varlıklarının yeniden kullanımı ile geliştirme için gerekli iş gücü azalmaktadır. Yazılım Ürün Hattı mimarisinin sağladığı ortak bileşen ve servisler birçok farklı ürün tarafından da kullanıldığı için yazılım kalitesi artmaktadır.

Anahtar Kelimeler: Yazılımın Yeniden Kullanımı, Yazılım Mimarileri, Yazılım Ürün Hatları, Yazılım Ürün Hattı Mühendisliği, Alan Mühendisliği, Servis Tabanlı Mimari, Dinamik Yazılım Ürün Hatları

ABSTRACT

USING SOFTWARE PRODUCT LINE ENGINEERING APPROACH FOR SOFTWARE DEVELOPMENT

Recep ATAŞ

Department of Computer Engineering

MSc. Thesis

Advisor: Prof. Dr. Oya KALIPSIZ

In recent years, as a consequence of software commodization, a major part of a software system is not unique for that software but also used by many similar software systems. For software systems like this, organizations should focus on the specific parts of the software rather than the common parts to develop softwares more effectively. Therefore, software reuse should be applied systematically for the software systems that share common software assets. There are so many concepts defined for software reuse research area to perform software development processes on a lower cost, faster and in quality basis.

All the new approaches on software engineering aims at enhancing the software reuse and productivity. One of the newest approach on this purpose is Software Product Line Engineering discipline, which is becoming to take an important share at software development and accepted by the software organizations. The idea of the Software Product Lines (SPL) relies on developing similar software products by reusing and configuring the common assets of these products and managing the variabilities between them.

The application domain of enterprise Alternative Distribution Channel (ADC) software products taken in this study, contain many software systems share common features which can be run on different distrubition platform. By considering this point, the study aims to show the success of the approach on a Software Product Line architecture exploited for developing J2EE and web based software systems within a

case study. The SPL case exploited in the study is a Dynamic SPL which provides configuring some variabilities at runtime by combining the SPL and Service Oriented Architecture approaches. For similar ADC applications, it is seen that Dynamic SPL approach improves the software productivity. Within exploiting the requirements, design, implementation and test artifacts in many products, the required software development effort is reduced and software quality is increased.

Key words: Software Reuse, Software Arcihtectures, Domain Engineering, Software Product Lines, Software Product Line Engineering, Service Oriented Architecture, Dynamic Software Product Lines

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Günümüzde organizasyonlar, yazılım sistemlerini daha hızlı geliştirmeyi, değişen müşteri gereksinimlerine daha hızlı cevap verebilmeyi ve üretkenliği artırmayı amaçlamaktadırlar. Bunun için yazılım geliştirme süresince yapılan işleri tekrar etmemek için yazılım mühendisleri, benzer yazılım sistemlerindeki ortak noktaları yeniden kullanıma göre hazırlamak isterler. Böylelikle önceden geliştirilen yazılım varlıklarını sonraki uygulamalarda yeniden kullanarak hızlı, kaliteli ve düşük maliyetli uygulamalar elde etmiş olmayı amaçlamaktadırlar. Bir yazılım sistemini oluşturan parçaların büyük bölümü sadece o yazılıma özgü değildir, benzer birçok yazılım ürünü tarafından da kullanılmaktadır. Büyük bölümü ortak olan yazılım sistemleri için yazılımın yeniden kullanımını sistematik olarak ele alınması gerekmektedir.

Yazılımın Yeniden Kullanımı (YYK), yeni bir yazılım sistemi oluştururken önceden var olan kaynak kod başta olmak üzere, gereksinim analizi, tasarım ve test durumları gibi yazılım varlıklarının tekrar kullanımıdır. Yeniden kullanım ilk olarak geliştirme aşamasında mevcut yazılımların kullanımı ile başlamıştır. Yazılımın yeniden kullanılabilirliği konusunda yapılan çalışmalara bakıldığında 1960'lar alt rutinler, 1970'lerde kod parçaları, 1980'lerde nesneler, son 20 yılda ise Nesneye Dayalı Programlama mantığı ile geliştirilen birçok teknik ve en son olarak Alan Mühendisliği ve Yazılım Ürün Hatları bu kapsamda kullanılmaktadır. YYK, organizasyonlar tarafından bir projenin belirli bir kısmında veya tamamında tercih edilebilmektedir. Yeniden kullanımın büyüklüğüne göre; nesne veya yöntemlerin yeniden kullanımı, yazılım

öğelerinin yeniden kullanımı ve tüm uygulama sisteminin yeniden kullanımı olarak düşünülebilir. Uygulamanın yeniden kullanılması tüm uygulamayı içerir, yani uygulama farklı projelerde tamamıyla yeniden kullanılabilir (COTS ürünleri gibi). Burada son yıllarda ortaya çıkan yaklaşımlardan biri, belirli ortak özellikleri olan sistemleri kapsayan Yazılım Ürün Hatlarıdır.

YÜH, Alan Mühendisliği veya diğer adıyla Yazılım Ürün Hattı Mühendisliği – YÜHM (Software Product Line Engineering - SPLE) olarak da literatürde yer bulmuştur. Ürün Hattı Mühendisliği gerçekleştirilirken belirli ortak özellikleri olan ürünler için ürün hattının kapsamında değişken ve ortak olan yazılım varlıkları bir araya getirilerek yeniden kullanımın arttırılması amaçlanır. Mimarinin sağlaması gereken en önemli özellik yazılım ürünlerinin farklılık gösterdiği yerlerde gerekli esnekliği sağlamaktır. Bunun için birçok değişkenlik yönetimi yaklaşımı sunulmuştur. YÜH, oyun yazılımları, silah sistemi yazılımları, web siteleri, mobil uygulamalar, gömülü yazılımlar, ERP uygulamaları gibi birçok farklı alanda kullanılmaktadır.

1.2 Tezin Amacı

Çoğu YÜH yaklaşımında ürünler arasındaki değişkenlikler yazılım sistemi gerçek ortama yayımlanmadan, yani çalıştırılmadan yönetilir. Yazılım sistemi yayımlandıktan sonra YÜH mimarisi tarafından sağlanan bir değişken ürün için seçilememektedir. Platform tipi, donanım tipi, işletim sistemi gibi değişkenliklere sahip olan uygulama alanlarında bu problem fazladan yazılım geliştirme iş gücü gerektirmektedir.

Bu problemi çözmek için YÜH ile ilgili olarak son yıllarda yapılan çalışmalar, değişkenliğin dinamik olarak yönetilebilmesi amaçlayan Dinamik Yazılım Ürün Hatları üzerinde yoğunlaşmıştır. Çalışmada bu yaklaşıma göre geliştirilmiş, üç katmanlı mimarilerde sunum katmanı için platform çoklayıcı bir model sunulmuştur. Bu modele uygun olarak geliştirilen bir YÜH ile ADK uygulamaları için farklı istemci tiplerinde çalışabilen yazılım sistemlerinin gerçeklenmesi gösterilmiştir.

Çalışmanın uygulama kısmında kurumsal ADK uygulamaları için yazılım üretkenliğini artırmayı amaçlayan bir yazılım ürün hattı mimarisi kullanılmıştır. ADK uygulamaları, farklı dağıtım kanallarında çalışabilen benzer uygulamalar olduğundan dağıtım kanalı

değişkeni dinamik olarak yönetilerek bu uygulamalarda YÜHM yaklaşımının kullanımının getirdiği faydaların gösterilmesi amaçlanmıştır.

1.3 Hipotez

Yazılım Ürün Hattı Mühendisliği yaklaşımı farklı ürün aileleri için kullanılmıştır. Benzer ürünler arasındaki değişkenliği yönetirken birçok farklı yöntem kullanılmıştır. Büyük ölçekli kurumsal ADK yazılım sistemlerinin geliştirilmesine bakıldığında ürünler arası benzerliklerin çok olduğu kadar ürüne özgü kısımlarında çok olduğu görülmüştür. Bu nedenle tek başına Yazılım Ürün Hattı Mimarisi kullanılması yeniden kullanımı artırmada bir fayda sağlasa da yeterli görülmemiştir. Çalışmada ürünler için değişkenlik yönetimi sağlamak ve yeniden kullanımı artırmak için geliştirilen Servis Tabanlı Mimari ile birlikte çalışan bir Yazılım Ürün Hattı mimarisi anlatılmıştır. Bu mimari kullanılarak Alan Mühendisliği ve ürün geliştirme adımlarının nasıl gerçeklendiği gösterilmiştir. Ayrıca ürünler arasındaki istemci tipi değişkenin dinamik olarak seçilebilmesi ile gerçeklenen bir sistem farklı istemci tipleri için çalışabilen masaüstü, web, web servis ve rapor için çalışabilmektedir. Böylelikle yazılım üretkenliği ve yazılım kalitesi büyük ölçüde artmaktadır.

BÖLÜM 2

YAZILIMIN YENİDEN KULLANIMI

Yazılım geliştirme, bir endüstri olarak söz edilmeye başlandığından bu yana araştırmacılar ve endüstride çalışan insanlar, hız, maliyet ve kalite iyileştirmelerini sağlayacak yöntemler ve araçlar üzerinde araştırmalar yapmaktadırlar. Bunu yapmak için gerekli olan yaklaşım oldukça nettir; bir kere geliştir, birçok defa kullan. Böylelikle yeni bir problemi çözmek için analistler, yazılım mimarları, geliştiriciler ve testçiler daha önce yapmış oldukları etkinlikleri tekrar yapmamış olacaklardır. Bu da yazılım üretkenliğini arttıracaktır. Günümüzde de birçok yazılım sistemi daha önceden gerçeklenmiş sistemlerden edinilen tecrübeleri, bilgiyi ve kaynak kodu kullanarak geliştirilmektedir.

Yazılımın Yeniden Kullanımı (YYK), yeni bir yazılım sistemi oluştururken önceden var olan kaynak kod başta olmak üzere, gereksinim, analiz, tasarım ve test durumları gibi yazılım varlıklarının tekrar kullanımıdır.

Tanım 2.1 Ezran'a göre [1] YYK, yazılım sistemleri geliştirilirken önceden oluşturulmuş blokların sistematik olarak kullanılmasıdır. Böylelikle uygulamalar arasındaki mimari ve gereksinimlerden faydalanılarak üretkenlik, kalite ve iş performansı konularında yarar sağlanabilir.

Yeniden kullanılabilirlik, bir yazılım elemanının yeniden kullanım olasılığını gösteren bir nitelik, *yeniden kullanılabilir varlıklar* da yeniden kullanılabilen bir yazılım veya yazılım bilgisi olarak tanımlanır [2].

YYK, yazılımın karmaşıklığını azaltarak hızlı, güvenilir ve düşük maliyetli ürün geliştirmek için yazılım mühendisliği alanında üzerinde çalışılmış ve çalışılmaya devam eden bir

konu başlığı olmuştur. Yazılım mühendisleri tarafından bir projenin belirli bir kısmında veya tamamında, yeniden kullanım mümkün olduğunca artırılmak istenmekte ve sistemler yeniden kullanıma göre tasarlanmaktadır.

YYK, her zaman yazılım geliştirmenin içerisinde olmuştur. [2], [3] de değinilen, YYK konusunda yapılan çalışmalara bakıldığında; bu kavramının 1960'larda alt rutinler, 1970'lerde kod parçalarıyla ortaya atıldığı ve Nesneye Dayalı Programlama, Bileşen Tabanlı Geliştirme gibi yeni yöntem ve teknolojilerin gelişmesiyle nesneler, bileşenler, vb. ile daha kapsamlı ele alındığı görülmektedir. Son yıllarda alana özgü yazılım mimarileri yeniden kullanım alanındaki önemli çalışma konularının başında gelmektedir.

Yeniden kullanımı sistematik olarak ele alabilmek için gereken başlıca işlevler soyutlama, seçme, özelleştirilebilme ve bütünleştirme olarak sıralanabilir [4]. Soyutlama; yeniden kullanılacak yazılım varlığına anlaşılabilir bir isim verilmesi, seçme; bu yazılım varlığının nasıl seçileceği bilgisidir. (Örn; Web servislerin tanımı için kullanılan WSDL dosyaları). Özelleştirme ise bu yazılım varlığının parametrik olarak çalışabilme yeteneğidir. Seçme ve özelleştirme birleştirerek komple bir yazılım sistemi oluşturulmuş olunur.

Bir yazılım ürününü oluşturmak için gereken birbirleriyle ilişkili görevler topluluğu *süreç* olarak tanımlanır. Bir organizasyonda etkinliklerin yönetiminin belirlenmesi açısından süreçler önemli ve gereklidirler. Her süreçte çalışan insanın diğerinde çalışan insandan bağımsız olacağı şekilde ayrı parçalardan oluşan alt süreçlere bölünebilirler. Yazılım süreçleri yazılım yönetimi ve üretilmesi görevlerinin hepsini kapsar. Yeniden kullanım süreçleri de bu aşamada yazılım varlıklarının geliştirilmesi ve tekrar kullanımı görevlerinin tanımlanması için gereklidir.

Bir yazılım yeniden kullanım süreci iki temel etkinlikten oluşur. Bunlar:

- Yeniden kullanım için geliştirme
- Yeniden kullanım ile geliştirme

Literatürde yeniden kullanılabilir yazılım geliştirmek için en etkili yolları bulmak için birçok çalışma yapılmıştır. Bu çalışmalar ileride detaylı olarak değineceğimiz iki konu üzerinde yoğunlaşmıştır. Bunlar; alan mühendisliği ve yazılım ürün hatlarıdır.

2.1 Yararları

C.R.U.I.S.E.'e göre [4] sistematik olarak yeniden kullanımın bir organizasyona faydaları şu şekilde sıralanabilir:

Yazılım Kalitesi Açısından: YYK, üretkenlik, güvenilirlik ve kalite iyileştirmeleri sağlar.

Kalite; her yeniden kullanımda varsa hatalar düzeltileceğinden, sadece bir kere geliştirilmiş ve kullanılmış bir bileşene göre daha yüksektir.

Üretkenlik; kazancı geliştirilen kodun daha az olmasından dolayı daha fazladır. Bu, aynı zamanda daha az test, analiz ve tasarım iş gücüne ihtiyaç duyulduğunu gösterir, toplam maliyeti düşürür.

Güvenilirlik; daha önceden kullanılmış ve test edilmiş bileşenlerin kullanımı yazılımın güvenilirliğini arttırmaktadır.

İş Gücü Azaltma; geleneksel yazılım geliştirmedeki her aşamada geliştirme zamanında azalma görülmektedir.

Bir Ürünün Piyasaya Sürülme Süresi (Time-to-Market); bir yazılım sistemin başarılı olup olmadığının önemli göstergelerinden biridir. YYK ile bu süre azaltılabilir.

Çalışan Sayısı; Artan yazılım üretkenliğiyle daha küçük takımlarla ürünler geliştirilebilir.

2.2 Yöntemler

Yazılım geliştirmenin başlangıcından günümüze kadar Yazılım Ürün Hatları dışında yeniden kullanım için birçok farklı yöntem geliştirilmiştir. Bir Yazılım Ürün Hattı mimarisi geliştirilirken bu mevcut yöntemlerden faydalanılmaktadır. Bu yöntemlerden bazıları aşağıda sıralanmıştır.

Nesneye Dayalı Programlama (Object-Oriented Programming - OOP): Yazılım geliştirmeye getirdiği yeni anlayışla yeniden kullanım için uygun bir geliştirme ortamı sunmuştur. Ancak OOP tek başına, yazılım sisteminin bütünü için sınıflar arasında tam

bir yeniden kullanım mantığı oluşturamamaktadır. Bunun için son yıllarda birçok ilgi çekici yöntem ortaya atılmıştır.

Mimari Kalıplar: Bir yazılım sisteminin yapısını, davranışını ve önemli özelliklerini göstermek için yüksek seviyeli soyutlama sağlayan yeniden kullanılabilen yapılardır (SOA, DSSA, vb). Genelde yazılım mimarileri bir bileşen kümesi, bu bileşenlerin entegrasyonu ve yapılandırılmasından oluşur [5].

Tasarım Kalıpları: Gang of Fours [6] tarafından sunulan bu kavram ile geliştirme sırasında karşılaşılan bir tasarım problemine önceden bulunmuş ve uygulanmış çözümlerin yeniden kullanımı kabul görmüştür.

COTS Ürün Yeniden Kullanımı: Bir COTS ürünü, kaynak kodları değiştirilmeden farklı müşterilerin ihtiyaçlarına uyarlanabilen yazılım sistemidir [7]. Bu tanım düşünüldüğünde yapılandırılabilen her yazılım sistemi bir COTS ürünü olarak ele alınabilir. Yazılım sisteminin tamamının yeniden kullanıldığı yaklaşımlardan biri olan bu yaklaşım son 15 yılda sıklıkla kullanılmaktadır. Örneğin, bir iş çözümü sistemi, yazılım gerçekleme hızını artırmak için OOP yerine COTS kullanımını tercih etmektedir.

COTS kullanımının getirdiği büyük faydalar olduğu gibi büyük riskleri de vardır. Kurum için doğru COTS ürünün seçimi zorlu bir süreçtir. COTS ürünleri bir daha değiştirilmesi zor olan kabuller üzerine geliştirilmiştir. Bunun için COTS kullanımında ileride olabilecek değişiklikler risk oluşturmaktadır. COTS kullanımının birçok başarı hikâyesi olduğu gibi başarısız uygulamaları da sıkça görülmektedir.

En çok kullanılan COTS örneği Kurumsal Kaynak Planlama, ERP, yazılım sistemleri olarak göze çarpmaktadır. [8] de Torchiano ve Morisio'ya göre açık kaynaklı ürünlerin kullanımı bir COTS kullanımıdır.

İlgiye Dayalı Yazılım Geliştirme (Aspect Oriented Software Development): Bir uygulama içerisindeki farklı ilgilerden (aspect) erişilebilen ortak bileşenleri derleme anında birbirleriyle bağlayarak yeniden kullanım sağlayan bir yöntemdir. Burada ilgi kavramı bir müşteri gereksinimi olarak düşünülebilir. Bu yaklaşımı sağlayan birçok uygulama ara yüzü mevcuttur (AspectJ gibi) [9]. Temel amaçları göz önünde bulundurulduğunda yeniden kullanım ile doğrudan ilişkili bir yazılım geliştirme

yöntemidir. Yazılıma kod seviyesinde esneklik ve yapılandırılabilirlik sağlamak için ilgileri ayırmayı amaçlar.

Alana Özgü Diller ve Kod Üreteçleri: Yazılım üretkenliğini ve yazılım kalitesini artıran hem analiz hem geliştirme süreçlerinde kullanılan bir yöntemdir. İleride detaylı olarak ele alınacaktır.

Sınıf Kütüphaneleri: Belirli bir amaç için geliştirilen sınıfların ara yüzlerini sunan kod kütüphaneleridir.

Yazılım geliştirmede bahsedilen ve ileride anlatılacak bu yeniden kullanım yaklaşımlarından hangisinin kullanılacağı seçilirken yani yeniden kullanımı planlanırken bir takım etkenlerin göz önüne alınması doğru karar vermek için faydalı olacaktır. Bunlar[7];

- Yazılım sisteminin geliştirme periyodu
- Yazılım beklenen yaşam süresi
- Geliştirme ekibinin tecrübesi ve yetenekleri
- Yazılımın kritik olup olmadığı ve fonksiyonel olmayan gereksinimler
- Yazılımın içinde bulunduğu uygulama alanı

Bu yöntemlerin dışında çokça kullanılan yazılım geliştirme yöntemlerini daha detaylı olarak incelemek yeniden kullanım yöntemlerini anlamak için önemli görülmüştür.

2.2.1 Bileşen Tabanlı Yazılım Geliştirme

Yazılım mühendisliği ve yazılım endüstrisi büyük sistemleri oluştururken daha önceden hazırlanmış bileşenler kullanarak üretkenlik ve kaliteyi artırma fikrini benimsemiştir. Bu fikir bilgisayar bilimlerinde çoğu alanda kullanılan böl ve birleştir yöntemi ile uyuşmaktadır. Bu kapsamda ortaya atılan Bileşen Tabanlı Yazılım Geliştirme (Component Based Software Development - CBSD), 1990ların başında ortaya çıkmış yeniden kullanımı artıran yöntemlerden biridir. Bileşenler, çalışma anında mevcut bir yazılım sisteminin çalışmasını etkilemeden yeni özelliklerin eklenmesini sağlayan bağımsız yazılım birimleridir [10]. CBSD, bu bileşenleri kullanarak ekle ve çalıştır mantığıyla çalışır. Bu yapı yazılım sistemlerinin bakımı sürecinde büyük faydalar

sağlamaktadır. Yeni gereksinimleri karşılamak için yeni bileşen eklemek veya var olan bileşenleri değiştirmek en uygun çözüm olarak görünmektedir.

CBSD ile ilgili bir kaç genel kuralı şu şekilde sıralayabiliriz:

Ara yüz tabanlıdır; yani bileşenler ara yüzü ve uygulamayı birbirinde ayırır ve uygulamanın detayını gizleyerek soyutlamayı sağlar.

Mimari tabanlıdır; yani daha önceden CBSD'ye göre tasarlanmış bir mimaride çalışabilirler, böylelikle yeni bileşen kolaylıkla diğer bileşenlerle birlikte çalışabilmektedir. Kullanılan mimariler genellikle MFC (Microsoft Foundation Class), CORBA, EJB gibi yapıların üzerinde kurulmaktadır.

Bileşenler, ürüne özgü, alana özgü veya alan bağımsız olabilmektedir (CORBA Domain Objects gibi). [4] de bileşenler için sınıflandırma şu şekilde yapılmıştır:

- GUI bileşenleri; en çok bulunan bileşenlerdendir. Bu bileşenlerde kod karmaşıklığı az olduğu için geliştirme maliyeti düşüktür. Örnek olarak; butonlar, metin alanları, veri gösterme için özelleştirilmiş tablolar gibi kullanıcı ara yüzleri verilebilir.
- Servis bileşenleri; daha karmaşık ve maliyeti yüksek bileşenlerdir. Uygulamaların ihtiyaç duyduğu ortak servisleri, veri tabanı erişimini sağlarlar. Örnek olarak; veri tabanı işlem servisleri verilebilir.
- Alana Özgü Bileşenler; geliştirmesi ve tekrar kullanımı en zor olan bileşenlerdir. Bileşenin kullanılacağı alan iyi tanımlanmış olmalıdır. Üretkenlik artışı çok fazladır. Örnek olarak; bir sigorta şirketi uygulaması için fatura, poliçe gibi bileşenlerin kullanılması verilebilir.

Geleneksel yazılım geliştirme yöntemlerinden farklı olarak CBSD'de yazılım geliştirme süreci ikiye ayrılır: Bileşen Geliştirme ve Bileşen Entegrasyonu [7]. Bu ayrıma göre bileşenlerin uygulanabilirlik, genel bir bileşen olma, diğer bileşenlerle birlikte çalışabilirlik gibi özellikleri incelenir.

Bir bileşen bir sisteme eklenmeden önce bazı uyarlamaların yapılması gerekebilir. Buna bileşenin optimizasyonu adı verilmektedir [4]. Bu işlem yapılırken bileşene yapılan değişiklik sonucu bileşenin temel özellikleri etkilenmemelidir. Örnek olarak; bir menü

bileşenin mobil cihazlarda daha küçük boyutta ve çözünürlükte gösterilmesinin menü bileşeninde bir düzenleme yapılarak sağlanabilir. Birbiriyle etkileşim içinde olan iki bileşen birleşerek yeni bir bileşen oluşturabilir. Buna bir bakıma uygulama çatısı da diyebiliriz.

Heineman'a göre [10], bileşenlerin birbiriyle etkileşimi istemci/sunucu ve yayıncı/abone mimarileri ile sağlanabilmektedir. İlkinde bileşenler bir istemci gibi davranarak istediği bilgiyi dönen yöntem çağrımı yaparlar. İkincisinde ise bir bileşene kayıt olup ondan sürekli bildirimler alırlar. Bir bileşen modeli bu sayılan özellikleri gerçeklemiş olmalıdır.

Bahsedilen seçme, kullanma ve entegrasyon zorluklarına rağmen bu yaklaşım sıkça kullanılmaktadır.

2.2.2 Uygulama Çatıları

Genel bir yapıya sahip belirli bir alt sistem veya uygulama için kod seviyesinde yüksek soyutlama getirerek yeniden kullanım sağlayan soyut ve somut sınıflar kümesidir[3]. Tasarım kalıplarını kullanarak iyi tasarım örnekleri sunarlar. Dile özgüdürler, yani Java, C, C++, Ruby, vs gibi dillerden sadece bir tanesi kullanılmıştır.

Uygulama çatıları kullanılarak oluşturulan uygulamalar, Yazılım Ürün Hatları ile ileri derece yeniden kullanımın temelini oluşturabilirler [4], [7]. Uygulama çatıları, alan bağımsız veya alana özgü olabiliyorken, ileride bahsedeceğimiz Yazılım Ürün Hatları yaklaşımı alana özgüdür.

2.3 Alana Özgü Yazılım Mühendisliği

Alana Özgü Yazılım Geliştirme, (Domain-Specific Software Engineering - DSSE) konusu literatürde genel adıyla Alan Mühendisliği olarak da geçmektedir. Belirli bir uygulama alanı için yeniden kullanılabilecek yazılım varlıklarının yazılım sistemleri geliştirmedeki geçmiş tecrübelerin biriktirilmesiyle elde edilip, yeni yazılım sistemleri geliştirirken kullanılmasını sağlar [11]. Alana Özgü Yazılım Mühendisliği 3 temel elemandan oluşur:

- Alana göre geliştirilmiş referans mimari
- Yeniden kullanılabilir bileşen kütüphanesi

- Bileşenleri seçme ve ayarlama için bir uygulama yapılandırma yöntemi

Referans mimari; bir uygulama alanı içindeki yazılım sistemleri için uygulanabilen ve yazılım sistemlerinin değişkenlik gösterdikleri notları barındıran temel özellik kümesidir[11]. Referans mimariler tek bir üründen türetilerek elde edilebilirler ve çoğu zaman tüm değişkenlikleri ve ortak noktaları içermezler.

Alana özgü mimariler, zamanla referans mimari, uygulama çatısı, bileşen geliştirme ve son olarak Ürün Hattı Mühendisliğinin temelini oluşturmuştur. Ürün hattı mimarileri ile alana özgü mimari kavramları birbirlerinin yerine kullanılabilmektedir. Aralarındaki fark alana özgü mimaride değişkenlik ve ortak noktalarının tamamlanmamış olmasıdır. Ürün Hattı mimarilerinde ise ortaklıklar ve değişkenlikler önceden belirlenmiş durumdadır.

Alana özgü mimarilerin uygulama geliştiriciler tarafından kullanılarak yazılım üretkenliğinin artırılabilmesi için Alana Özgü Dillere ihtiyaç duyulmaktadır.

Alana Özgü Diller ve Modelleme: Belirli bir uygulama alanı içerisinde yer alan ortak özellikleri olan yazılım sistemlerini geliştirirken alana özgü dillerden faydalanılabilir. Burada sözü geçen uygulama alanlarına örnek olarak web tabanlı eğitim uygulamaları, çevrimiçi bilet rezervasyon satış sistemleri, finansal uygulamalar vb gösterilebilir. Bir DSL ile yazılım geliştirmede, kodun büyük bölümü DSL kod üreticileri tarafından otomatik olarak üretildiği için daha hızlı, kolay ve kaliteli olacaktır.

[12], [13] deki tanımlara göre Alana Özgü Dil, belirli bir problem uzayına göre ayarlanmış bir tanımlama dili veya yüksek seviyeli bir programlama dili olarak düşünülebilir. Alana özgü modelleme ile bu terim daha çok popüler olmaya başlamıştır. Bu dillere örnek olarak bir çağrı merkezi uygulama geliştirme dili, istatistik uygulamaları için kullanılan R ve S dilleri, HTML, SQL, Regular Expressions, HQL, Eclipse Modelling Language gibi birçok programlama ve tanımlama dili verilebilir. Java, C, C#, UML gibi diller ise genel amaca yönelik programlama ve tanımlama dilleri olduğu için DSL'nin tam tersi olarak düşünülebilir.

Alana özgü diller [12];

- Tecrübeli problem alanını iyi bilen yazılım mimarları tarafından geliştirilir.
- Kendi kendilerinin dokümantasyonunu yaparlar.

- Genel amaçlı dillere göre daha fazla soyutlama sağladığı için kalite, üretkenlik, bakım yapılabilirlik ve tekrar kullanılabilirlik artmaktadır.

Yukarıdaki avantajları olduğu gibi dezavantajları da vardır. Bunlardan bazıları; yeni bir dil öğrenme maliyeti, bu dilin sağladığı özelliklerin bazen kısıtlı olması, yazılım mimarları alanın gereksinimleri için yeteri kadar etkin bir dil geliştirilememiş olması olarak sayılabilir.

DSL'in gerçeklenmesinde yani koda dönüştürülmesinde iki yöntem izlenebilir. Birincisi kod oluşturma, ikincisi çalışma zamanında DSL'den alınan bilginin yorumlanarak uygulanması şeklinde açıklanabilir.

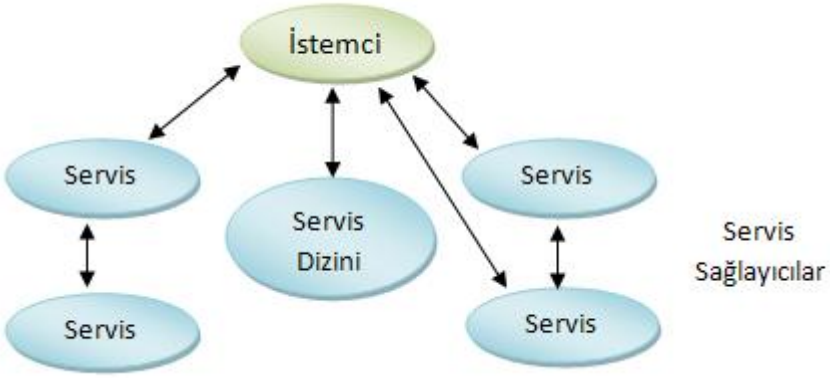
Alana özgü dillerin büyük çoğunluğu metin tabanlı olduğu gibi grafik ara yüzüne sahip diller de mevcuttur. Grafik ara yüzlü dillerin sistematik olarak kullanımına *Alana Özgü Modelleme* adı verilmiştir. Alana özgü modelleme örnek olarak Eclipse tarafından geliştirilen Eclipse Modelling Framework[1] verilebilir.

2.4 Service Tabanlı Mimari

Servis Tabanlı Mimari, uygulama sistemleri için belirlenmiş bir işi yerine getirmek için bloklar sunan yeniden kullanılabilir ve belirli standartlara sahip bir yazılım mühendisliği yaklaşımıdır [7]. Servis ara yüzleri XML tabanlı bir dil olan WSDL ile tanımlanabilir. Bir WSDL belirtimi, ara yüzün türünün ve sağladığı işlemlerin tanımlarını, servis tarafında kullanılan bağlama protokolünü ve servisin bulunduğu yeri yani URL'ini içerir. Servis ara yüzleri organizasyon için daha sonradan birçok farklı uygulama ve istemci tarafından kullanabileceği bir yazılım varlığıdır. Bu sebeple yeniden kullanımı büyük ölçüde artırmaktadır.

Şekil 2.1'de SOA'nın elemanları ve bu elemanlar arasındaki iletişim gösterilmektedir. Buna göre bir istemci servis dizininde olduğu sürece farklı sağlayıcıların sunduğu servisleri çağırabilmektedir.

[1] Eclipse Modelling Framework, EMF, www.eclipse.org/emf



Şekil 2.1 Servis Tabanlı Mimari genel görünümü

SOA ve sonraki bölümlerde değineceğimiz YÜH yazılım geliştirme yaklaşımları benzer amacı paylaşırlar. Her ikisi de organizasyonların varlıklarını ve yazılım geliştirmedeki olgunluklarını tekrar kullanmalarını sağlar. Bu yaklaşımlar yeniden kullanımı aktifleştirerek üretkenlik artışı, düşük geliştirme maliyeti, yüksek güvenilirlik gibi bir üründen beklenilen özellikleri karşılamaktadırlar.

SOA, değişen gereksinimlere hızlı bir şekilde cevap verebilmek için kurumsal çözümlerin toplanması, düzenlenmesi ve bakımını kolaylaştırır. SOA kullanılan bir uygulamada yeniden kullanılabilirlik üst seviyededir. Oluşturulan her servis farklı projelerde dahi olsa yeniden kullanılabilmektedir. İş gereksinimlerinde bir değişiklik olduğunda bu değişikliğin uygulama tarafındaki etkisi sadece bu serviste olur. Yani ilgili değişiklik sadece bu serviste yapılır ve bu servisi kullanan uygulamalar değişiklikten büyük oranda etkilenmez.

YÜH, belirli müşteri veya pazar için oluşan değişkenlikleri yönetirken, birbirleriyle ilişkili bir dizi sistem arasındaki ortak noktaları sistematik olarak yakalar ve bunları işler. Bu amaçlar doğrultusunda SOA ile SPL arasında bir bağ olduğu ve servislerin YÜHleri desteklediği söylenebilir.

2.5 Ürün Hattı Mimarilerine Doğru

Yazılım geliştirmede zaman, kalite ve maliyet gibi konularda iyileştirme sağlamak için tekrar kullanabilen kabuller oluşturmak önemlidir. YYK'nın da amacının yazılım üretkenliğini arttırmak olduğunu söylemiştik. Buna göre Yazılım Mühendisliği

kapsamında ortaya atılmış OOP, CBSD gibi önceden bahsedilen teknolojilerin yanı sıra Servis Tabanlı Mimari (Service-Oriented Architecture - SOA) ve Yazılım Ürün Hattı Mimarilerinin (Software Product Line Arcihectures) kullanımının getirdiği faydalar kabul görmüştür. 2000'li yıllardan sonra YYK adına en farklı fikir, yazılım ürün hatları kavramı, SPL ile gelmiştir [14]. 90lı yılların sonuna kadar daha önce bahsettiğimiz Alan Mühendisliği yaklaşımları ortaya atılmıştı, fakat Bayer'e göre bunların beklendiği kadar etkili bir yeniden kullanım sağladığı ispatlanamamıştır [4]. Bunun temel sebebi olarak da uygulama alanın kapsamının tam olarak tanımlanamaması, operasyonel bilgi eksikliği gösterilmiştir.

Yazılım Ürün Hatları, benzer yazılım sistemlerini, bu sistemler tarafından paylaşılan ana kabulleri, bir başka tabirle yazılım varlıkları kullanarak geliştirmeyi amaçlayan bir yöntemi ifade etmektedir. Organizasyonlar, ortak bir fabrikayı kullanacak benzer ürünlerin üretim mekanizmasını oluşturmak için birçok yöntem kullanmıştır. Mesela; otomotiv şirketlerinde bir bölüm otomobillerde kullanılacak parçaları üretirken diğer bir bölüm bu parçaların çeşitli kombinasyonlarda kullanımı ile çeşitli otomobiller üretebilmektedir.

Yazılım ürün hattı mimarileri daha önce bahsettiğimiz birçok yeniden kullanım yöntemini barındırır. YÜHlerin bu yöntemlerden bazıları ile olan ilişkisi aşağıda listelenmiştir.

- Alan mühendisliği kapsamında Alana Özgü Dil kullanımı ile yeniden kullanımı artırmaktadır.
- Ürün hattı sahip olduğu ürünlere göre değişebilen özellikler özellik ağaçları ile gösterilebilir.
- Servis Tabanlı mimari ile birlikte ele alındığında yeniden kullanımı arttırdığı gözlemlenmiştir.

2.5.1 Öznitelik Odaklı Yazılım Mühendisliği

Kyo Kang [15], yazılımın yeniden kullanımı ile ilgili birçok girişim olduğunu, fakat bunların çoğunun sadece iki yönde yapıldığını söylemiştir. Bunlar; alana özgü yazılım mimarileri, bileşen entegrasyonu ve uygulama üretme mekanizmaları üzerine deneysel

araştırmalar ve yazılım mimarileri, mimari tanımlama dilleri, yeniden kullanılabilir varlıkların geliştirilmesi konuları üzerine teorik araştırmalardır. Ortak noktaları tespit etme ve bu bilgi üzerinde yeniden kullanım için çalışma konusunda az çalışma yapıldığını göz ününde tutmuştur. Buradan yola çıkarak Feature-Oriented Reuse Method (Öznitelik Odaklı Yeniden Kullanım Yöntemi), FORM geliştirilmiştir [4].

FORM, özellik (feature) olarak adlandırılan, bir alan dâhilinde yer alan uygulamalar arasındaki ortak noktaları ve farklılıkları tespit edip alan mimarisi ve bileşenleri geliştirmeye dayalı bir yöntemdir. FORM'da özellikler müşteri ile mühendisler arasında ürünün sahip olduğu veya sağladığı özellikler olarak düşünülür. Gereksinim ve fonksiyonlar özellik ismi ile tanımlanırlar. Özellikler sağladıkları fonksiyonellikle birbirlerinden farklıdırlar.

FORM yöntemi Alan Mühendisliği ve Uygulama Mühendisliği olarak iki ana süreç içerir. Alan Mühendisliği 3 aşamadan oluşmaktadır; İçerik Analizi, Alan Modelleme, Mimari Modelleme. Modelleme kısmı özellikler ile sağlanır. Mimari ise modeller kümesinden oluşan ve farklı sistemlerde mimari oluşturmak için kullanılan bir referans modeldir. Uygulama mühendisliği aşamasında geliştirilen özellikler kullanılır.

Özellik Tabanlı Yazılım Mühendisliği (Feature Oriented Software Engineering), ürün hattı mimarilerindeki özellikleri göstermede kullanılmaktadır [15], [16]. Burada özellik, yazılım geliştirme sırasında elde edilen her bir artı değer olarak düşünülebilir. Bir yazılım ürün hattından çıkan her bir ürün, ürün hattı sağladığı özelliklerin farklı birleşimlerinden oluşmuştur. Ürün hattını sağladığı değişkenlik ve değişim noktaları özellik tabanlı modelleme ile bir özellik ağacında gösterilebilmektedir.

Bir ürün hattı mimarisindeki özellikler 3 tipte olabilir. [15] de belirtilen özellik tipleri aşağıda açıklanmıştır.

Zorunlu özellik; tüm ürünler bu özelliği içermelidir.

Seçimli özellik; her ürün olması zorunlu olmayan özelliklerdir.

Değişken özellikler; ürünlerin değişkenlik gösteren gereksinimlerine göre ayarlanabilen özelliklerdir.

2.5.2 Yazılım Ürün Hattı Mühendisliği

Alan Mühendisliğinin belirtilen kısıtlamaları ile zayıf ürün hattı yaklaşımlarından yola çıkarak Bayer tarafından Yazılım Ürün Hattı Mühendisliği, YÜHM önerilmiştir. Atkinson'a göre [17] PuLSE (Product Line Software Engineering), farklı amaçlar farklı içeriklerde başarı ile kullanılabilir.

PuLSE, üç temel elemandan oluşmaktadır; konuşlanma (deployment), teknik bileşenler ve destek bileşenleri. Taşıma aşaması bir yazılım ürün hattının alt yapısının yüklenmesi, başlatılması ve yönetimini içerir. Teknik bileşenler, ürün hattı geliştirmeyi sağlayacak teknik bilgi sağlar. Destek bileşenleri, ürün hattı taşınması, geliştirilmesi, adaptasyonu için bilgi sağlar.

Ürün Hattı mühendisliği üzerine yapılan çalışmalardan KobrA yaklaşımı [17], ürün hatları, bileşen tabanlı geliştirme, uygulama çerçeveleri ve süreç modelleme gibi yazılım mühendisliği teknolojilerin bir sentezi olarak Atkinson tarafından ortaya atılmıştır. Alan planlama ve alan kapsamı tanımları yapılmıştır. İki aşamadan oluşur. Framework Engineering, tüm ürün farklılıklarını temsil eden ve ortak noktaların bilgilerini içeren genel bir uygulama çerçevesi oluşturur. Uygulama mühendisliği aşamasında belirli bir uygulama yaratmak için bu uygulama çerçevesi kullanılır.

Çalışmanın bir sonraki bölümünde Yazılım Ürün Hattı Mühendisliği yöntemi Yazılım Ürün Hatları ve mimarileri olarak detaylı olarak incelenmiş ve bu disiplin için geçerli tanım çerçevesi aktarılmıştır.

BÖLÜM 3

YAZILIM ÜRÜN HATLARI

Günümüzde yazılımın yeniden kullanımı konusunda en etkili yaklaşımlardan ve üzerinde en çok araştırma yapılan konulardan biri Alan Mühendisliği ile birlikte Yazılım Ürün Hatlarıdır. Özellikle geliştirme zamanını düşürmesi ve üretkenliği arttırdığının gösterilmesiyle bu yaklaşımın kullanımı yazılım geliştirmede kabul görmeye başlamıştır.

SEI' ye göre bir Yazılım Ürün Hattı, belirli bir alan üzerinde düşünen veya belirli bir görevi yerine getiren ürünler kümesidir [18].

Sommerville'in yaptığı tanımlamada [7] Yazılım Ürün Hattı, her biri gereksinimlerine göre özelleştirilmiş uygulamalardan oluşan ortak mimari ve bileşenlere sahip ürünler olarak geçmektedir. Ana sistem, farklı müşterilerin ihtiyaçlarına uyacak şekilde ayarlanabilir ve uyarlanabilir olmalıdır.

Üretim alanında yeni bir yaklaşım değildir. Bugüne kadar otomotiv, uçak, bilgisayar, mobil cihazlar gibi birçok alanda ürün hatları geliştirilip kullanılmıştır. Ürünler arasındaki ortak noktalarını temel alan Yazılım Ürün Hatları, önemli ve geçerli bir yazılım geliştirme yaklaşımı olarak hızla yükselen yeni bir fikirdir.

Ürün esnekliği yazılım pazarı için büyük öneme sahiptir. Ürün hatları, müşteri veya müşteri gruplarının ihtiyaçlarına özel sistemler oluşturabilmeyi sağlamaktadır. Bunu yaparken ürünlerin ortak noktalarından oluşan varlıkları kullanarak üretkenliği artmasını sağlarlar.

Yazılım ürün hattı mühendisliği yaklaşımını benimseyen ve buna uygun mimariler geliştiren organizasyonlar dikkate değer miktarda üretkenlik, ürün kalitesi, hız, müşteri memnuniyeti gibi konularda ilerleme sağlamışlardır [18],[19],[20]. Yazılım ürün hattı

yaklaşımının sağlayabileceği faydalara çalışmanın sonraki bölümlerinde detaylı olarak değinilecektir.

Bunların yanında bu yaklaşımının bir takım riskler doğurabileceğini de söylemeliyiz. Böyle bir yaklaşım kullanımı organizasyonu teknik olarak yeni bir geliştirme yöntemine adapte olması demektir. Organizasyon ve teknik yönetim açısından karmaşık mühendislik iş gücü gerektirmektedir. Alan tecrübesi olmayan organizasyonlar için başarısızlıkla sonuçlanabilir.

Ürün hattı yaklaşımını benimseyen organizasyonlar;

- Ürünlerinin yapısı
- Misyon veya Pazar
- İşletme amaçları
- Organizasyon yapıları
- Personelin teknik yeterliliği
- Olgunluk ve mevcut alan ile ilgili bilgi birikim ve tecrübelerine göre geniş bir alanda değişmektedir.

Buna rağmen, ortak varlıklar kümesinden yeni ürünler oluşturmada gerekliliği evrensel olarak kabul edilen aktiviteler ve çalışmalar mevcuttur. Bu çalışmada, ürün hattı geliştirmek için tanımlanmış bir çerçeve detaylı olarak açıklanmıştır.

3.1 Yazılım Ürün Hattı Tanımı ve Kavramı

Clement'e göre [14] Yazılım Ürün Hattı, belirli bir pazara yönelik ortak ve yönetilmiş özellikleri olan yazılımların, ortak yazılım varlıklarının yeniden kullanımı ile geliştirildiği sistemler topluluğudur.

"A Software Product Line (SPL) is a set of software intensive systems, which share a common, managed set of features that satisfy the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way."

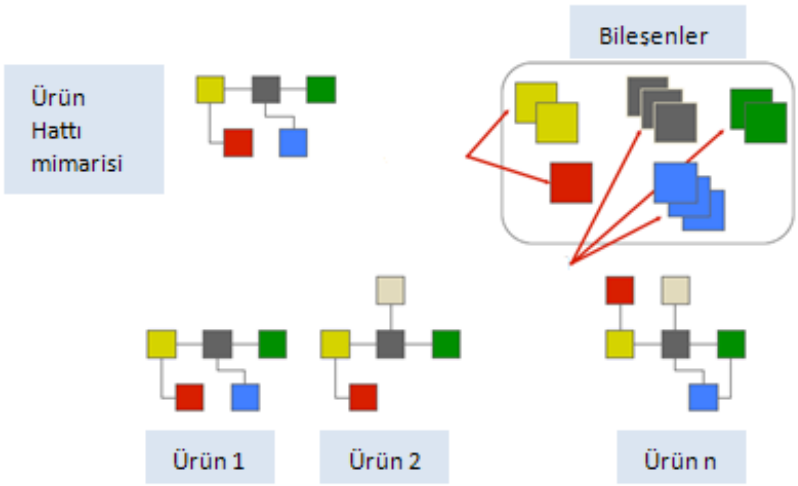
Yazılım Ürün Hatlarını, Bileşen Tabanlı Geliştirme ile kıyaslarsak birinde bileşenlerin yeniden kullanımı diğerinde ise mimarinin yeniden kullanımı söz konusudur.

Yazılım Ürün Hattı Mühendisliği, PLSE; çoğu çalışmada Yazılım Ürün Hatları kavramı yerine kullanılmaktadır ve YÜH ile aynı prensiplere sahiptir. Bu yaklaşım bir ürün ailesi doğrultusunda yeniden kullanılacak yazılım varlıklarının geliştirilmesini ve yeni yazılım sistemleri geliştirirken bu kabullerin tekrar kullanımını amaçlar. PLSE'nin süreçleri ve aktiviteleri çalışmanın bir sonraki bölümünde detaylı olarak ele alınmıştır.

Bir Yazılım Ürün Hattı, genellikle bir ürün hattı mimarisine sahiptir. Bir ürün hattı mimarisindeki ortak varlıklardan uygulamaya uygun olan bileşenler seçilip düzenlenerek yeni ürünler oluşturulur. Bunun için mimari gereken seviyede parametrik ve genişletilebilir olmalıdır. Her ürün bu referans mimariyi türeterek, bileşenleri yapılandırarak ve kendisine özgü kısımları geliştirerek kendi mimarisini oluşturur [11].

Genel bir yazılım mimarisi tek bir yazılım sisteminin yapısını tanımlarken, bir ürün hattı mimarisi birbiriyle ilintili ürünlerin mimari yapılarını tanımlar.

Ürün Hattı mimarilerinin genel amacı, program ailelerinin tasarım ve bakımının yapılmasını kolaylaştırmaktır. Bir ürün hattı mimarisi, yeniden kullanımı en üst seviyeye çıkarmak için tüm ürün mimarilerinde bulunan ana varlıkları, belirli ürün mimarilerindeki farklılıkları gösteren değişim noktalarından ayırır.



Şekil 3.1 Yazılım Ürün Hattı mimarisi

Bir ürün hattı mimarisi kullanarak geliştiriciler sadece ürüne özgü problemlerle uğraşırlar. Şekil 3.1'de YÜH mimarisinin genel yapısı gösterilmiştir. Her yazılım

ürününün mimarisi ana mimariden seçilerek oluşturulmaktadır. Yeni bir uygulama geliştirme süresince iş gücünün büyük çoğunluğu programlamadan ziyade entegrasyondur. Her Yazılım Ürün Hattında ürün geliştirmek için tanımlanmış bir kılavuz veya plan olmalıdır.

Aşağıda Yazılım Ürün Hattı Mühendisliği için kullanılan terimler yer almaktadır.

Ana Varlık (Core Asset): Yazılım ürün hattının temelini oluşturan yeniden kullanılabilir varlıklar ve kaynaklardır. Ana kabuller, mimari, yeniden kullanılabilir yazılım bileşenleri, alan modelleri, gereksinimler, dokümantasyon, bütçe, test durumları, iş planları, süreç tanımlarını içerir ve bu liste daha da genişletilebilir. En önemli kısım daha önce de bahsettiğimiz gibi mimaridir.

Alan (Domain); belirli bir uzmanlık alanındaki bilginin özelleştirilmiş halidir [21]. Örnek olarak telekomünikasyon alanı anahtarlama, ağlar, protokoller ve telefonlardan oluşur. Bir telekomünikasyon yazılım ürün hattının bu fonksiyonları sağlaması beklenir.

Kapsam (Domain Scope); Belirli bir alanın içerisine hangi ürünlerin alınabileceğini gösterir.

Yazılım Ürün Hattının Uygulanması; bir yazılım ürün hattından ürün elde etmek için ana kabullerin sistematik olarak kullanılmasıdır.

Referans Mimari (Reference Architecture); belirli bir alana özgü bileşenleri ve özellikleri barındıran, bu alan kapsamındaki ürünleri geliştirirken referans alınan mimaridir [11].

Bunların dışında literatürde ürün hattı yerine ürün ailesi, platform, alan mühendisliği yerine ana kabul geliştirme, uygulama mühendisliği yerine ürün geliştirme gibi Yazılım Ürün Hatları ile ilgili terimler kullanılmaktadır.

Değişkenlik (Variability); alan mühendisliği sırasında tanımlanır, uygulama mühendisliği sırasında uygun değişkenler seçilerek kullanılır [22].

Değişim Noktası (Variation Point); ürünlerin nerelerde farklılaştığını belirtir [22]. Bu noktalarda uygulama geliştirici bir karar vererek uygun değişkenlerden birini seçer. Her değişim noktası bir veya birden fazla değişkeni seçenek olarak sunar.

Değişken (Variant), bir değişim noktasına karşılık gelen, ona uyan her bir seçeneğe, özelliğe değişken adı verilir.

Özellik (Feature); bir yazılım sisteminde bir grup fonksiyonel veya kalite gereksinimleri bütünüdür. Bu özellikler YÜH'nin kullanıcılarına veya geliştiricilerine görünebilir durumdadır. Bir YÜH'de ortaklıklar ve değişkenlikler ürün özellik ağacı ile gösterilebilir.

Yazılım Ürün Hatları tam olarak anlamak için bir yazılım ürün hattı mimarisine tam olarak karşılık gelmeyen diğer yeniden kullanım yaklaşımlarından bahsetmek faydalı olacaktır.

[14],[18] de SEI için yazılan YÜH çerçevesinde Clement, Yazılım Ürün Hatlarının;

- Geçici veya az miktarda kodun yeniden kullanımı,
- Yeniden kullanım ile tek bir yazılım sistemi geliştirme,
- Sadece bileşen tabanlı veya sadece servis tabanlı yazılım geliştirme,
- Sadece yapılandırılabilen bir mimari,
- Ürünlerin yeni sürümleri,
- Birkaç tane teknik standart olmadığını belirtmiştir.

3.2 Yazılım Ürün Hatları ve Uygulama Çatıları

Yazılım Ürün Hatları ve önceki bölümlerde bahsedilen Uygulama Çatılarının birçok ortak yönü vardır. İkisi de ortak bir mimari ve bileşenleri içerirler ve yeni bir yazılım sistemi oluşturmak için bunları kullanırlar. Bu iki yaklaşım arasındaki temel farklar ise aşağıdaki şekilde sıralanmıştır [7].

- Uygulama Çatılarında genişletme yapmak kalıtım, yerine kullanma, aşırı yükleme gibi OOP temellerine dayanır. Genellik kaynak kod değiştirilmesine izin verilmez. YÜH'de ise OOP gibi bir disipline bağımlılık ve limit yoktur. Bileşenler istendiği gibi seçilip, değiştirilebilir ve yenileri eklenebilir.
- Uygulama Çatıları belirli bir alana özgü değil genel olarak teknik kod desteği verir. Örneğin, web tabanlı uygulama geliştirmeye yarayan uygulama çatıları bulunur.

YÜH ise daha detaylı alan ve platform desteği içerir. Örneğin hastane kayıtları için web uygulamaları geliştirilebilen bir ürün hattı olabilir.

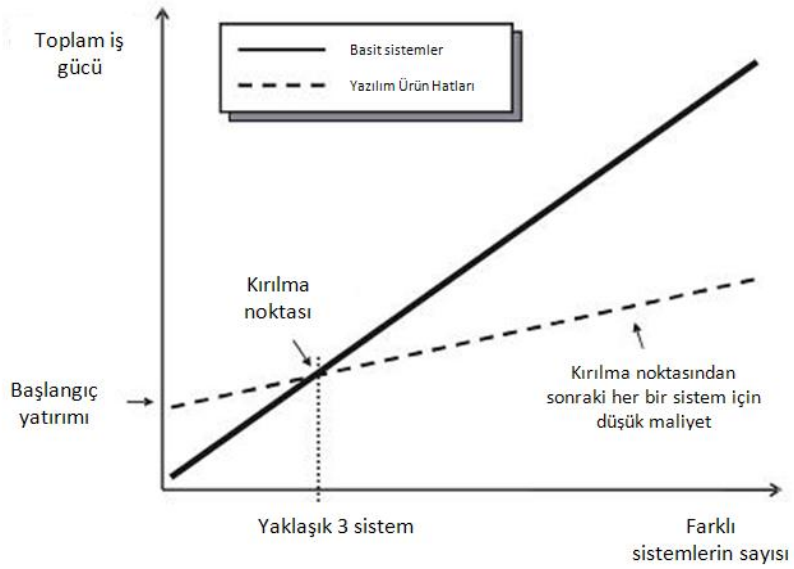
- YÜHler genellikle donanım, işletim sistemi, istemci tipi gibi yazılım sistemi farklılıklarını desteklerken, uygulama çatıları sadece yazılım tabanlıdır.

Eğer bir organizasyon OOP tabanlı bir YÜH geliştiriyorsa, Uygulama Çatılarını bu iş için temel alabilir [4]. Uygulama çatısını genişleterek alana özgü hale getirebilir.

3.3 Yazılım Ürün Hatlarının Maliyeti ve Faydaları

SEI'nin YÜH yaklaşımını kullanan organizasyonlarla yaptığı araştırmada aşağıdaki sonuçlar çıkmıştır [18].

- Yüksek ölçekli üretkenlik kazancı
- Ürün geliştirme hızının artması
- Yazılım kalitesinin artması
- Daha sağlıklı maliyet tahmini yapabilme
- Kod sayısının azalması
- Ürün risklerinin azalması
- Müşteri memnuniyeti



Şekil 3.2 Yazılım Ürün Hattı üretim kazancı [19]

Şekil 3.2'de Yazılım Ürün Hatlarının sağladığı kazanç grafikle gösterilmiştir. [19] a göre YÜH kullanımı geliştirilen ürün sayısı arttığında mühendislik iş gücü bakımında daha fazla fayda sağlamaktadır.

Ana kabuller geliştirildikten sonra her ürün geliştirme aşamasında kazanç elde edilebilmesi için aşağıdaki maddelerle [18] ilişkilidir.

Gereksinimler: Ürünlerin ortak gereksinimleri ürün hattının sağladığı gereksinim katmanını oluşturur. Geniş kapsamlı gereksinim analizi yapılmış olmalı ve fizibilite tamamlanmış olmalıdır.

Mimari: Bir yazılım sisteminin mimarisi tecrübeli yazılım mühendisleri tarafından uzun bir sürede de hazırlanır. Her ürün için ürün hattı mimarisinin doğru çalışan bir örneği oluşturulabilmelidir.

Bileşenler: Ana kabullerdeki bileşenlerin hepsi ürünlerde kullanılabilecek şekilde parametrik ve genişletilebilir olmalıdır. Beklenen değişkenlik noktaları dikkate alınacak ve performans sorunu oluşturmayacak şekilde genel olmalıdır.

Modelleme ve Analiz: Ürün için geçerli gereksinim analizi ve tasarım modelinden oluşur. Performans modeli ve ilgili analizler bir ürün hattının ana kabullerindendir. Her yeni ürün için performans sorunu ve uygulama hataları giderilebilmelidir. Mimari ile bir bütün halinde alana özgü bir dil ile desteklenmelidir.

Test: Ürünlerle ilgili değişkenlik için test durumları bulunmalıdır.

Planlama: Ürün geliştirme planlaması yapılmış olmalıdır. Geçmiş ürün geliştirme bilgileri ile bir iş planı çıkarılabilir.

Süreçler: Yapılandırma yönetimi araçları ve genel yazılım geliştirme süreçleri yer almalıdır.

İnsan: Ürün geliştirmek için daha az insan gerekmektedir. Çalışanlar kolaylıkla bir başka ürüne kaydırılabilirler. Çalışanlar genel yazılım mühendisliği eğitimi almış olmalı ve ana kabulleri kullanabilmelidirler. Yeni çalışmalar için eğitim dokümanları bulunmalıdır.

YÜH, getirdiği bu faydalara göre ana kabullerin geliştirilmesi sırasında büyük miktarda bir başlangıç yatırımı doğurmaktadır. Ürün hattı yaklaşımını kullanmak isteyen

organizasyonlar bu yatırımı yapmalıdırlar. Bunun yanında ürün hatları için büyük yatırımlar yapmanın her organizasyon için uygun olmayabileceğini de düşündüğümüzde yazılım evlerinin bu yaklaşımı kullanması daha olasıdır.

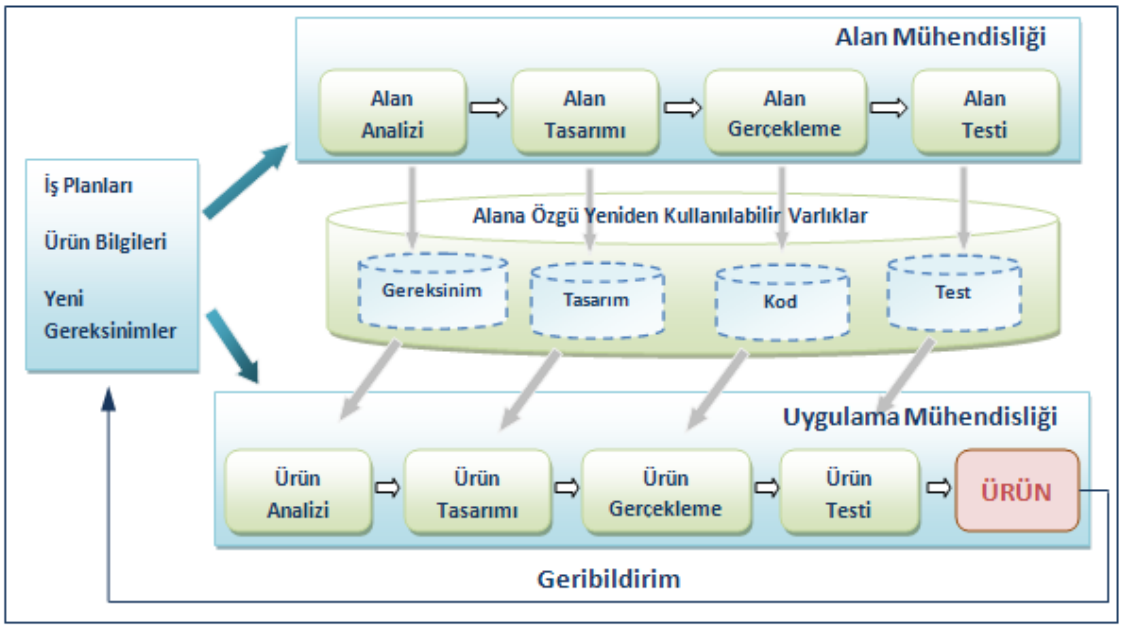
3.4 Temel Yazılım Ürün Hattı Mühendisliği Etkinlikleri

Temelde, bir ürün hattı ana varlık geliştirme ve bu ana varlıklarla ürün geliştirmeyi içermektedir. Yazılım Ürün Hatlarında 3 temel etkinlik bulunmaktadır [14].

Alan Mühendisliği; tekrar kullanılacak ve yapılandırılabilecek ana varlıkların ortaklıklarının ve değişkenliklerinin tanımlanmasını ve geliştirilmesini kapsar.

Uygulama Mühendisliği; ana varlıklar kullanılarak ve yapılandırılarak ürünlerin geliştirilmesini kapsar.

Yönetim; organizasyonun bu etkinlikler içindeki süreçleri yönetimini kapsar.



Şekil 3.3 Yazılım Ürün Hattı süreçlerinin birbirleriyle ilişkileri

Şekil 3.3'de YÜHM süreçlerinden Alan Mühendisliği ve Uygulama Mühendisliği alt süreçlerinin birbirleriyle olan ilişkileri gösterilmektedir. Her iki süreç arasında bulunan alana özgü yeniden kullanılabilir varlıklar, Alan Mühendisliğinin çıktısı ve Uygulama Mühendisliğinin girdisi durumundadırlar. Geleneksel yazılım geliştirme yöntemindeki adımlar YÜH'lerde alan ve uygulama mühendisliği adımlarında ayrı olarak ele

alınmaktadır, ancak uygulama mühendisliğindeki süreçler alan mühendisliğindeki süreçlere bağımlıdırlar. Bu iki aktivitenin sırası karışık ve tekrarlı olabilir. Yani, ilk önce ana varlıklar geliştirip bu ana varlıklarla ürünler geliştirme ya da mevcut ürünlerden ana varlıkları elde etme şeklinde olabilir.

Uygulama Mühendisliği adımında yeniden kullanılabilir yazılım varlıkları ile ilgili geribildirim yapılabilir. Geribildirim mevcut yazılım varlıklarının ihtiyaçlara göre değiştirilmesi, yeni özellikler eklenmesi veya yeni yazılım varlıklarının eklenmesi gibi olabilir.

Şekil 3.4'de SEI tarafından belirlenen aktiviteler arası ilişki gösterilmektedir. Ürünler ile ana varlıklar arasında sürekli bir geri dönüşüm söz konusudur. Yeni ürünler geliştirildikçe ana varlıklar yenilenir. Ayrıca ana varlıkların yeni geliştirilebilecek ürünleri göz önünde bulunduracak şekilde genel bir yapıda olmaları sağlanır. Yönetim genel olarak ürün hattı ile geliştirilebilecek ürünleri tespit etmeye çalışır. Bunun için yeni ürünlerin ürün hattı ile bağlantılı olmaları ya da ürün hattının bu yeni ürünleri kapsaması sağlanır. Bir sonraki bölümde bu etkinlikler daha detaylı olarak incelenmiştir.

Şekil 3.4 Ürün Hattı Mühendisliği aktiviteleri [18]

3.4.1 Alan Mühendisliği ile Ana Varlık Geliştirme

Ana Varlık Geliştirmenin (Core Asset Development) amacı ürün hattına üretim yeteneği kazandırmaktır. Ortaklıkların ve değişkenliklerin tanımlanması, ürünlerin paylaştığı

ortak yazılım varlıkların geliştirilmesini içerir [24]. Bu aşamada, değişim noktaları tasarlanır ve her bir değişim noktasıyla ilişkili değişkenler gerçeklenir [19].

Alan Mühendisliğinde;

Analiz: Yeninden kullanım için ürünler arasındaki ortaklık ve değişkenliğin belirlenmesi ve ürün hattı gereksinimlerinin analizini içerir. Buna **alan kapsamı**, yani YÜH'nin ürünlerinin neler olabileceği belirlenir.

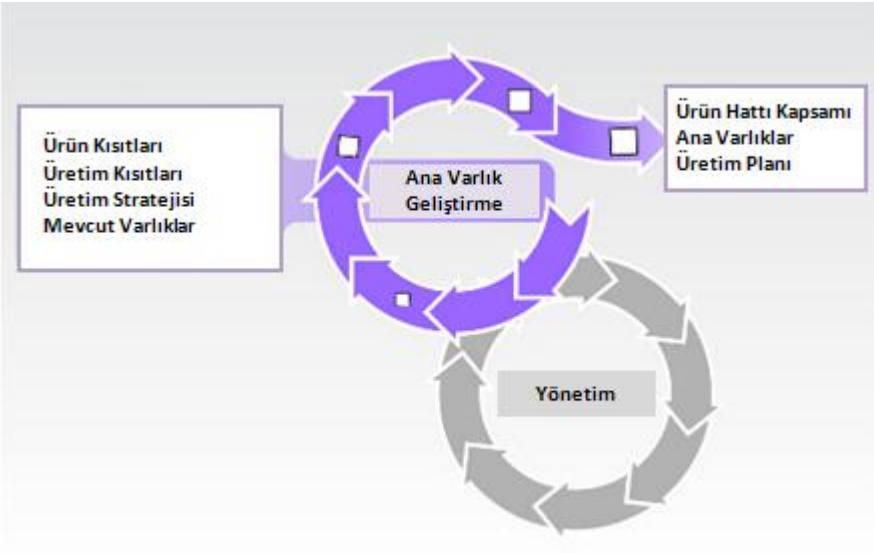
Tasarım ve Gerçekleme: Ürün Hattı mimarisinin ve bileşenlerin, değişim noktaları ile birlikte tasarım ve gerçeklemesini içerir [18]. Ürün hattından farklı ürünler elde edebilmek için gerekli alt yapı bu süreçte gerçeklenir.

Ürünün açılması ve bakımı: Üretim planı ve ürün yapılandırılmasını içerir.

Alan Mühendisliği sonrası çıktı olarak; yazılım bileşenleri, yeniden kullanılabilir ve ayarlanabilir gereksinimler, tasarımlar gibi yazılım varlıkları elde edilir.

3.4.1.1 Alan Mühendisliği Girdileri

Şekil 3.5'de bu aktivitenin çıktıları ve etkilendiği faktörler mevcuttur. Buradaki tekrarlı bir şekilde çalışan akış ana varlıkların geliştirilme yöntemine göre kimi zaman ok yönünün tersinde de işleyebilir. Buradaki SEI tarafından belirlenen Alan Mühendisliği girdi parametreleri detaylı açıklamaları aşağıda yer almaktadır [18].



Şekil 3.5 Ana varlık geliştirme [18]

Ürün Kısıtları: Ürünler arasında hangi ortak noktalar ve değişkenliklerin var olduğu, hangi özellikleri sağlamaları gerektiği, ileride faydalı olacak özelliklerin neler olabileceği, kurumlar, şirketler, müşterilere özgü uygulanacak standartları, performans limitleri, yazılım kalite gereksinimlerin, fiziksel kısıtlamaların neler olduğu sorularına cevap aranır [18]. Bir YÜH için bu kısıtlar, mevcut birkaç ürün üzerinden yorumlanarak elde edilebilirler. Bileşenlerin geliştirilmesi ve yazılım mimarisinin tasarımını bu kısıtlar etkiler.

Üretim Kısıtları: Ürünün piyasaya çıkma zamanı, ürünleri kim geliştirecek ve hangi ortamlarda çalışılacak, ürünleri oluşturmak için bir otomatik kod oluşturucu (generator) mı kullanılacak yoksa elle mi kodlama yapılacak gibi sorulara cevap aranır ve buna göre bir karar verilir. Ana kabullerdeki değişkenlik yönetiminin nasıl olacağı hangi yöntemlerin kullanılacağı buna göre belirlenir.

Üretim Stratejileri: Ana varlıkların ve ürünlerin tam olarak gerçekleştirildiği yaklaşımdır. YÜH'nin nasıl oluşturulacağı belirlenir. Burada iki yöntem vardır.

Proaktif (önceden hazırlanan); geliştirmeye ana varlıklar ile başlanır ve ürünler bu varlıklara göre gerçeklenir.

Reaktif (sonradan hazırlanan); bir grup ürün ile başlanır ve ana varlıkları oluşturmak için bu ürünlerin bileşenleri genelleştirilmeye çalışılır.

Bu iki yöntemin ikisi birlikte de kullanılabilir. Genel bileşenler geliştirilecek mi yoksa satın mı alınacak, bu bileşenlerin maliyetleri ürünlere nasıl dağıtılacak, ana varlık geliştirme nasıl yönetilecek gibi sorulara cevap aranır.

Üretim stratejileri buna bağlı olarak yazılım mimarisini ve bileşenlerini etkilemektedir. Aynı zamanda hangi ürünlerin geliştirilebileceği sürecini de yönetir.

Mevcut Varlıklar: Mevcut bileşenler, ürünler ve sistemler bir organizasyonun alan deneyimi ve market yapısının gövdesini oluşturur. Bir ürün hattı mimarisi daha önceden denenmiş kütüphaneleri, uygulama çatılarını, algoritmaları, geliştirme araçlarını, servisleri, bileşenleri veya mevcut ürünleri kullanabilir. Aynı şey yönetim ve finansman modelleri için de geçerlidir. Mevcut varlıklar organizasyonun kendi geliştirdiği varlıklarla sınırlı değildir, harici olarak uygun olan üçüncü parti COTS, web

servisler, açık kaynak ürünler, standartlar, tasarım kalıpları, uygulama çatıları gibi varlıkları da içermesi büyük avantaj sağlayacaktır.

3.4.1.2 Alan Mühendisliği Çıktıları

Alan mühendisliği sonunda elde edilen çıktılar organizasyonun yeniden kullanım planını göstermektedir.

Ürün Hattı Kapsamı: Bir ürün hattının içerebileceği ürünlerin tanımıdır. En basit haliyle kapsam ürünlerin isimlerinden oluşabilir. Bu tanım ürünlerin sağladıkları işlevler, sahip oldukları kalite özellikleri veya üzerinde çalıştığı platformdan oluşabilir. Bir ürün hattının başarılı olabilmesi için kapsamının doğru belirlenmesi gerekir. Kapsam çok genişse ürün değişkenliği çok büyükse, ana varlıkların değişkenliği yönetebilmesini zorlamış oluruz ve ürün hattı ile ürün geliştirme eski yönteme dönebilir. Yani yazılım geliştirme hiç ürün hattı kullanmıyormuş gibi zorlaşabilir. Eğer kapsam küçük olursa bu kez de, ana varlıklar yeterince genele özgü olmayabilir. İleride gelecek yeni bir geliştirme için cevap verilemeyebilir ve ürün hattının dinamikliği azalır. Bunlara bağlı olarak ürün hattına yapılan yatırımın getirisi az olur.

Bir organizasyonun YÜH yaklaşımında kapsam, organizasyonun hedeflerine uygun doğru ürünleri hedeflemelidir. Kapsam, organizasyonun planının değişmesi, piyasa şartlarının değişmesi, yeni iş alanlarının ortaya çıkması gibi durumlarda daha da gelişebilir. Örneğin mobil uygulamalarının günümüzde kullanımının artması ürün hattınızın kapsamını genişletmeniz için iyi bir sebeptir.

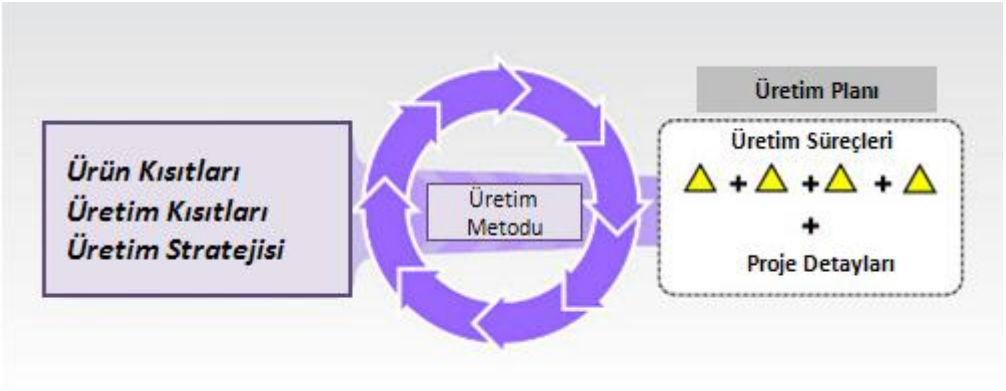
Ana Yazılım Varlıkları: Ana varlık tabanı üretimde temeli oluşturan tüm ana varlıkları içerir. YÜH kullanılarak geliştirilen bir üründe, tüm ana varlıkların kullanılması zorunlu değildir. Bu ana varlıklar, YÜH üstünden tüm ürünlerde sistematik olarak yeniden kullanılabilmek üzere bir yazılım mimarisi ve yazılım bileşenleri içerirler.

Test durumları, test planları, gereksinimler, alan modelleri, YÜH kapsamı, web servisler, COTS, kütüphaneler, uygulama çatıları da aynı zamanda ana varlıktırlar.

Yönetim sürecinde yer alan bütçe, plan, geliştirme temelini oluşturan Alana Özgü Diller – DSL, geliştirme araçları, otomatik kod üreticileri de ana varlıktır.

Tüm bu ana varlıklar ürün geliştirirken nasıl kullanılacağını tanımlayan birer yazılım geliştirme sürecine bağlı olmalıdırlar. Örnek olarak bir ürün geliştirmede gereksinimlerin hazırlanması sürecinde ilk önce ana varlıklardaki gereksinim kullanılmalıdır.

Üretim Planı: Ana varlıklardan nasıl üretim olacağını söyler. Yukarıda söylediğimiz her bir ana varlığın ait olduğu süreçler ve bunların birleştirilmesi üretim süreçlerini oluşturur. Bu üretim planı, Şekil 3.6'da gösterilen üretim yöntemini etkiler, yani modelin ve süreçlerin tanımlanması, ana varlıklara ait bu süreçlerde hangi araçların kullanılacağını belirler. Şekil 3.6'da ürün kısıtları, üretim kısıtları ve üretim stratejisinin de üretim yöntemini etkilediği gösterilmiştir.



Şekil 3.6 Alan mühendisliği çıktısı üretim planı [18]

Örneğin bir YÜH'de *değişkenlik*, bir bileşen ve servis kümesinde seçim yapılarak, yeni bir bileşen eklenerek veya çıkarılarak, bileşenleri kalıtım yolu veya parametrik yaparak özelleştirerek ya da ilgiler kullanarak gerçekleştirilebilir. Tüm bu yöntemler üretim metodu örnekleridir.

Ürünler arasında gerekli olan değişkenliği sağlayan metodun hangileri olacağı üretim sürecinde tanımlanır. Uygun olmayan bir varyasyon mekanizması seçmek üretim sürecinin verimini azaltır. Üretim planı aynı zamanda zamanlama, ücretler, metrikler gibi proje detaylarını da içerir.

Alan Mühendisliği çıktıları birkaç farklı şekilde özelleştirilerek Yazılım Ürün Hattından yeni bir yazılım ürünü gerçeklenebilmektedir. Bunlar [7];

- Platformun özelleştirilmesi: Uygulamanın farklı platformlar için farklı versiyonlarının gerçeklenebildiği Yazılım Ürün Hatlarıdır. Örneğin, bir ürünün farklı versiyonları Windows, Linux ve MacOS platformlarında çalışabilmesi verilebilir.
- Donanım özelleştirilmesi: farklı donanım yapılarına uygun ürünlerin oluşturulabilmesini sağlayan YÜHler bu kapsamdadır. Özellikle gömülü yazılımlar ve mobil uygulamalarda görülürler.
- Fonksiyonların özelleştirilmesi: Farklı müşteri gereksinimlerine göre ürünlerin özelliklerinin seçilebilmesini sağlayan YÜHlerdir.

3.4.2 Uygulama Mühendisliği ile Ürün Geliştirme

Ürün geliştirme aktivitesi yukarıda gösterilen ana varlık geliştirme aşamasının 3 çıktısına; ürün hattının kapsamı, ana varlıklar, üretim planına bağlı olarak yapılır [18],[19]. Ayrıca ürüne özgü tanımlamaların ve geliştirmenin yapıldığı ürün tanımlama alt süreci bulunmaktadır.

Yazılım ana varlıkları kullanılarak yeni bir ürün oluşturulur. Ürüne özgü yazılım kısımları geliştirilir. Uygulama mühendisliği aşamasında, değişim noktalarında ürünün ihtiyacına göre belirli değişkenlere bağlanır.

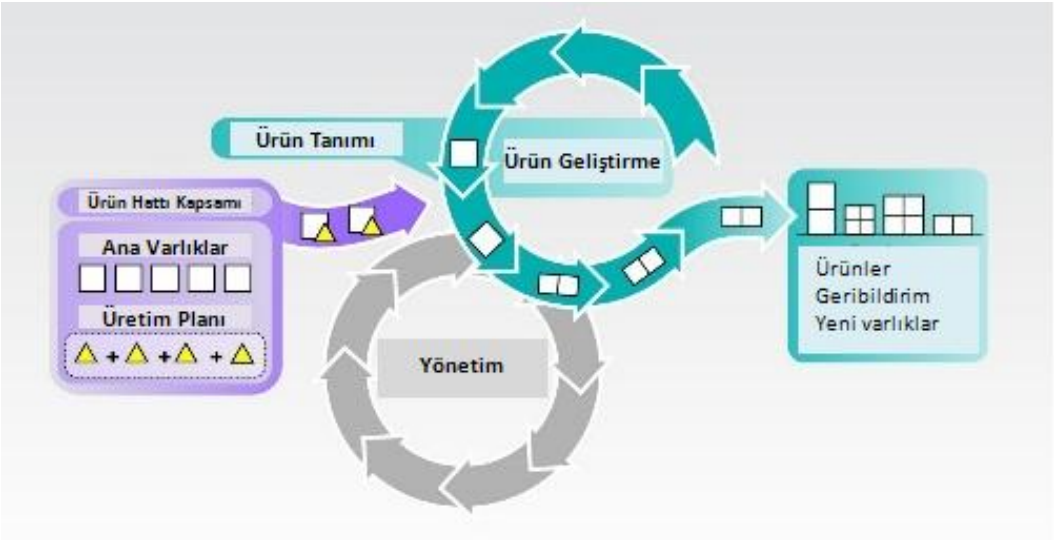
Şekil 3.7 ürün geliştirmedeki genel yapıyı göstermektedir. Örnek olarak belirli bir ürünün mevcut olması yeni bir ürün için gereksinimleri etkiler. Ortak yönleri daha önceden hazırlanmış ürünlere göre az olan yeni bir ürün gerçeklenmek istendiğinde ana varlıklar güncellenmelidir ve bu ortak nokta daha sonraki ürünlerde kullanılacak şekilde ana varlıklara eklenmelidir.

Ürün geliştirmeyi yönlendiren girdiler şu şekildedir:

- Ürünün tanımı; ürün hattı kapsamı içinde belirli bir genel tanımın varyasyonudur.
- Ürün hattı kapsamı, ürün hattının bu ürünü içerecek kadar esnek olup olmadığını belirler.
- Ana varlıklar, bu ürünün geliştirilmesinde kullanılacak ana varlıklardır.

- Üretim planı, ana varlıkların nasıl kullanılacağını detaylı olarak anlatır. Ürünleri geliştirenler aynı zamanda ana varlıklarla ilgili hata veya iyileştirme gibi geri dönüşlerde bulunurlar.

Böylelikle ürün hattının sağlamlığına ve kalitesine katkı sağlarlar.



Şekil 3.7 Ürün geliştirme [18]

Ürün geliştirme sürecinin çıktıları; yeni ürünler, geliştirme sırasında alan mühendisliği sürecine yapılan geribildirimler ve varsa yeni geliştirilen yazılım varlıklarıdır.

Sistemin belirli bir versiyonunu oluşturarak yeni bir ürün elde etmek için Alan Mühendisliğinin sağladığı bileşenler seçilerek ve parametreleri ayarlanarak özelleştirilir. Uygulama mühendisliği süresince yazılım sistemini gerçeklemek için aşağıdaki adımlar uygulanır.

- Yazılım sistemi için gereksinimler alınır.
- Bu gereksinimlere en uygun yazılım ana varlıkları seçilir. Seçilen bu yazılım varlıkları değiştirilmeye hazırdır.
- Ürüne özgü detaylı gereksinim analizi yapılır.
- Detaylı gereksinim analizine göre seçilen yazılım varlıkları yapılandırılır veya değiştirilir. Eğer ana varlıklarda gereksinimi tam karşılayan bir bileşen yoksa bu ürün için ek geliştirme yapılır.

3.5 Ortaklıların ve Değişkenliklerin Yönetimi

Bir Yazılım Ürün Hattı mimarisinin yerine getirmesi gereken en önemli özellik belirli ortaklıkları olan ürünler arasındaki değişkenliğin yönetilmesidir [22]. Bir yazılım ürün hattının ürünler arasındaki ortak varlıkları içermesi kadar ürünlerin farklılaştığı yerleri görüp buna göre bir mimari sunması da önemlidir [25],[26].

Ortaklıklar; ürün hattı kapsamındaki her ürün tarafından paylaşılan özellikler olarak kabul edilir.

Değişkenlik ise daha önce de bahsettiğimiz gibi bir yazılım sistemini ürüne göre düzenleme özelliğidir.

Değişkenlik yönetimi: Ürün Hattının alan kapsamındaki ürünler arasındaki farklılıkların yönetilmesidir. Bunu sağlamak için değişkenliğin tanımı, ürün hattı ile birlikte kullanımı ve değişken seçme zamanı belirtilmelidir.

Değişken Bağlama zamanı, değişim noktasında bir değişkenin seçilme işleminin ne zaman yapılacağını söyler. Bunlar;

- Mimarinin yapılandırılması, bileşen seçme zamanı
- Bileşen yapılandırılması, bileşene parametre geçme zamanı
- Geliştirme zamanı
- Yükleme zamanı
- Derlenme zamanı
- Açılış zamanı
- Çalışma zamanı

Değişkenlik yönetimini aktif hale getirmek için değişim noktaları ve değişkenlik modeli gerekir (Örneğin nesneye dayalı programlamadaki kalıtım).

Bir YÜH'de ortaklıklar ve değişkenlikler sınıflandırılmış gösterimi için ürün özellik ağacı kullanılabilir. Ortaklıklar ve değişkenlikler toplamda 4 tipte gösterilebilirler. Çizelge 3.1'de değişkenlik tipleri gösterilmiştir.

Çizelge 3.1 Özellik tipleri [27]

TİP	AÇIKLAMA
Zorunlu	Her üründe bulunması gereken fonksiyonlar, özellikler
Seçimli	Her üründe olması zorunlu değildir
Alternatif	Bir grup özellikten sadece bir tanesi seçilebiliyorsa, seçilenin tipi alternatiftir.
Çoklu Alternatif	Bir grup özellikten bir veya daha fazlası seçilebiliyorsa, seçilenleri tipi çoklu alternatiftir.

3.5.1 Değişkenlik Yönetimi Mekanizmaları

Bir yazılım ürün hattında değişkenliğin gerçeklenmesi zorlu bir işlemdir. Literatürde ve uygulamada birçok değişkenlik mekanizması fikri ortaya atılmış ve kullanılmıştır. Bunlardan bir kısmı genel yazılım geliştirme dâhilinde de kullanılan yöntemler olduğu gibi bir kısmı da ürün hattı mimarilerine özgü yöntemlerdir.

[14], [18], [28] de yer alan Jacobson'un belirttiği değişkenlik mekanizmaları;

- Kalıtım; sınıf tanımı sırasında, sınıfın gerçeklenmesi (örneğin Java'daki implements)
- Genişletilmesi; gereksinim sırasında, sınıfın genişletilmesi (örnek Java'daki extends)
- Sınıfın Kullanımı; gereksinim sırasında, bir sınıfın başka bir sınıfı kullanması
- Yapılandırma; çalışma zamanından hemen önce, bileşeni özelleştirmek için ayrı bir dosyadan bilgileri almak (XML, JavaBeans properties dosyası gibi)
- Parametre, bileşen gerçekleme sırasında, bir fonksiyonun parametrelerini bir sınıf olarak almak, bir bakıma yeniden düzenlemiş kod (calculate(Rule rule) gibi)

- Şablon Tanımlama, bileşen gerçekleme sırasında, bir sınıfın tipi belirli bir tanım dâhilinde olacak şekilde sayısız alt tipten oluşabilir. (Java'daki *generics* kütüphanesi bunu sağlar Örnek: ExceptionHandler<Container>)
- Üretim, çalışma zamanı sırasında ve daha önce, kullanıcının girdiği değerlere göre tanımları üreten bir araç (Yapılandırma sihirbazları gibi)
- Nesneye yönelik programlama tekniği olan delegasyon kullanımı; bir objenin yapamayacağı bir işi bu objeyi genişleterek elde edilen delegasyon objesi ile yapmak
- Aşırı Yükleme; Java'daki *overloading*
- Delphi dilindeki sınıfların özelliklerinin değiştirilebilmesi
- Java'daki sınıf üretme için yansıma (reflection); ürüne göre çalışma anında yüklenecek sınıflara karar verme
- Statik kütüphaneler, bir ürünü özelleştirmek için bu kütüphanelerin yeni yazılanla değiştirilmesi
- İlgiye Yönelik programlama
- Tasarım kalıpları, örnek olarak Adapter kalıbı
- *Property* dosyaları
- Dinamik sınıf yükleme

Bileşen geliştirmenin ürün hattı mimarileri için sağladığı birçok özelliğin dışında riskleri de bulunmaktadır. Çok fazla değişkenlik olması durumunda kod çok karmaşıklaşabilir, değişkenlik mekanizması doğru seçilmezse ilerde bileşeni yeniden yazmak gerekebilir. Bunun için değişkenliklerin ne olduğu, nerede ve nasıl olduğu, hangi değişkenlere sahip olduğu gibi bilgilerin dokümantasyonunun yapılması gerekmektedir.

3.5.2 Yönetim

Önceki bölümlerde anlatılan Alan mühendisliği ve Uygulama mühendisliği aktiviteleri dışında bir diğer önemli aktivite de yönetimdir. Yönetim, bu iki aktivitenin düzenli işlemesini sağlamalıdır. Yazılım ürün hattı yönetimi iki şekilde incelenmektedir [14].

Teknik yönetim; ana varlıklar ve ürün geliştirme aktivitelerinin birlikte çalışmasını sağlar. Üretim metodu kararını vermede yer alır ve üretim planı yönetimi ile ilgili elemanları sağlar.

İş yönetimi; müşteri ilişkileri, kaynak yönetimi, bütçe, ürün geliştirme ile ana varlık geliştirme arasındaki teknik aktiviteler, üretim kısıtları, ürün hattına adapte olma gibi işleri düzenler.

3.5.3 Aktiviteleri Bir Araya Getirme

Ana varlık geliştirme, ürün geliştirme ve yönetim ürün hattı mühendisliğinde ana aktivitelerdir ve iyi bir şekilde birleştirilmelidir. Farklı organizasyonlar bunu gerçekleştirmek için farklı yollar kullanabilir.

Birçok organizasyon yazılım ürün hattı hazırlamaya ilk önce ana varlıkları oluşturarak başlar (Proactive yaklaşım). Ürün hattını oluşturacak ürünlerin kapsamını tanımlarlar ve bu kapsam ürün hattının mimarisini, bileşenlerini, değişkenlik noktalarıyla diğer ana varlıkların tasarımında kullanılır. Buradaki **değişkenlik noktası** mimarinin ve bileşenlerin yapılandırılmasıdır.

Bazı organizasyonlar da mevcut ürünlerden yola çıkarak YÜH hazırlarlar (Reaktif yaklaşım). Her iki yönetiminde avantaj ve dezavantajları vardır. Proaktif yaklaşımda ilk önce önemli olan ana varlıkları geliştirilebilir. İlk ürünler bunları kullanır yeni ihtiyaçlar ortaya çıktıkça sonraki ürünlerde bunlar ana varlıklara eklenebilir. En sonunda planlanan tüm ana varlıklar gerçeklenmiş olur. Preactive yaklaşımda da buna benzer artan bir yöntem kullanılabilir.

Proaktif yaklaşımda ürünler çok fazla kod yazmadan hazırlanabilir. Dezavantajı ise bu yaklaşımda tüm ürün kümesi için genele özgü bir yapıda olacak mimari ve bileşenleri gerçekleştirmek iyi bir yatırım ve o alanla ilgili iyi bir bilgi ve tecrübe gerektirmektedir. Belirli bir alanda uzun süredir projeler geliştiren bir organizasyon için bu çok büyük bir dezavantaj değildir, ama eğer uygulama alanı ile ilgili tecrübe sahibi değilse büyük bir risk oluşturur. Reaktif yaklaşımın avantajı ise mevcut varlıkları kullandığı için maliyetinin daha düşük olmasıdır. Fakat YÜH'nin başarılı olması için kullanılan varlıkların yeterince esnek olması ve sonradan geliştirilecek ürünlerin ihtiyaçlarına

cevap verebilmesi gereklidir. Aksi takdirde bu varlıkları iyileştirmek daha maliyetli olabilir.

3.6 Kullanım Örnekleri

YÜHM yaklaşımı oyun yazılımlarından gömülü gerçek zamanlı yazılımlara, mobil cihazlardan web sitelerine kadar birçok farklı uygulama alanı için kullanılmıştır.

[29] daki çalışmada Ishida YÜH yaklaşımını kurumsal sistem geliştirme için kullanımından elde ettiği faydaları ve karşılaştığı sorunları göstermiştir. Ülkemizde yapılan YÜH çalışmalarından Aurora Yazılım Ürün Hattı ile yine birçok büyük ölçekli ve web tabanlı bankacılık ve sigortacılık programları, merkezi kayıt sistemleri gibi birçok kurumsal yazılım uygulaması geliştirilmiştir [30]. ASELSAN A.Ş. mühendisleri tarafından geliştirilen YÜH ile silah sistemlerinde kullandığı yazılımlar için hızlı ve kolay geliştirme sağlamıştır [23].

Benzer özellikte web portalleri için YÜHM yaklaşımının kullanımının mümkün olduğu gösterildiği çalışmada [31] bir YÜH mimarisinin bileşenleri gösterilmiştir.

Yazılım Ürün Hatları konusu üzerine uluslararası bir konferans olan SPLC tarafından belirlenen YÜH yaklaşımının nasıl olması gerektiğini gösteren önemli modellerden oluşan ünlü YÜH çalışmaları listesinde birçok büyük kurumun farklı uygulamalar için bu yaklaşımı benimsediği görülmektedir [32].

Ülkemizde bu alanda yapılan akademik çalışmalardan biri olan Parlakol tarafından sunulan tez çalışmasında [33] YÜHler için test tabanlı bir servis modeli anlatılmış ve bu modelin kredi kartı yazılımları için örnek kullanımı gösterilmiştir. [26] daki doktora çalışmasında özellik tabanlı YÜHleri ile ilgili bilgi ve örnek çalışmalar sunulmuştur.

BÖLÜM 4

PLATFORM ÇOKLAMA için SERVİS TABANLI YAZILIM ÜRÜN HATTI MODELİ

Yazılım Ürün Hattı Mimarileri ile Servis Tabanlı Mimarinin birlikte kullanıldığı birçok çalışma olmuştur. PLSE'de bir ürünün nasıl yapılandırılacağı özellikler ile gösterilirken, SOA'da servisler kullanılır. Ürünler paydaşlar arasında genellikle öznitelikler ile ifade edilirler. Ürün hattı mimarisindeki ortak noktalar ve değişkenlikler öznitelikler ile gösterilebilir. Bu gösterimden yola çıkarak referans mimari ve kaynak kod gibi ana varlıklar seçilir ve yapılandırılır.

SOA'nın en önemli özelliği yazılım sistemini bir servis olarak düşünmesidir. Diğer taraftan geliştirme süresini kısaltmak için *Alan Mühendisliği* çalışması sonucu geliştirilen ana varlıkları da kullanmamız gerekmektedir. Bu konu da SPLE'nin ilgi alanındadır.

4.1 Servis Tabanlı ve Dinamik Yazılım Ürün Hatları

SOA ve YÜH yazılım geliştirme yaklaşımları benzer amacı paylaşırlar. Her ikisi de organizasyonların varlıklarını ve yazılım geliştirmedeki olgunluklarını tekrar kullanmalarını sağlar [34], [35]. Bu yaklaşımlar yeniden kullanımı aktifleştirerek üretkenlik artışı, düşük geliştirme maliyeti, yüksek güvenilirlik gibi bir üründen beklenilen özellikleri karşılamaktadırlar.

SOA, değişen gereksinimlere hızlı bir şekilde cevap verebilmek için kurumsal çözümlerin toplanması, düzenlenmesi ve bakımını kolaylaştırır. Oluşturulan her servis farklı projelerde dahi olsa yeniden kullanılabilmektedir. Bu amaçlar doğrultusunda SOA ile SPL arasında bir bağ olduğu ve servislerin YÜHleri desteklediği söylenebilir.

Birçok PLSE yaklaşımının ana yazılım varlıklarından yapılandırılmış ürünler geliştirmeyi amaçladığını söylemiştik. Ancak birçok YÜH örneği, ürün hattı değişkenliklerinin ürünün gerçek ortama alınmadan (deployment) önce yapılandırılmasına odaklanmaktadır [36]. Yani, müşteriye teslim edilen yazılım sistemi, bileşenleri seçilmiş ve yapılandırılmış halde olmaktadır. Böyle olunca müşterinin bu yazılım ürününü çalıştırma sonrasında değiştirmesi çoğu zaman mümkün olmamaktadır.

Bu problemi aşmak için yazılım sisteminin taşınmasından sonra da dinamik olarak yapılandırılabilmesi gerekmektedir. Yazılım ürününe dinamik olarak ürün hattı özellikleri eklenebilmeli, çıkarılabilmeli ve değiştirilebilmelidir.

Örnek olarak yazılım sistemini, donanım veya işletim platformunu anlayarak dinamik olarak yapılandırabilme düşünülebilir. Mesela YÜH kullanılarak gerçeklenen bir sanal ofis uygulamasının dizüstü bilgisayar, PDA veya mobil telefondan da çalıştırılabilmeyi desteklemesi istenebilir. Burada alan kapsamına yeni cihazlar ve fonksiyonların eklenebileceği de göz önünde bulundurulduğunda bu iş için mevcut YÜH yaklaşımlarından daha fazlasına ihtiyaç duyulacaktır.

Bunu çözmek için araştırmacılar Dinamik Yazılım Ürün Hatları - DSPL fikrini ortaya atmışlardır [36].

Servis Tabanlı Yazılım Ürün Hattı, STYÜH, yaklaşımı servis mimarisini ve servisleri kullanan bir DSPL'dir [35], [36]. Bu yaklaşım son zamanlarda sıkça kullanılarak ve araştırma yapılarak kabul görmüştür. Sanal ofis örneğindeki farklı cihazlarda çalışabilme özelliği Servis Tabanlı Mimari ile sağlanabilmektedir [36].

4.2 Platform Çoklayıcı Model

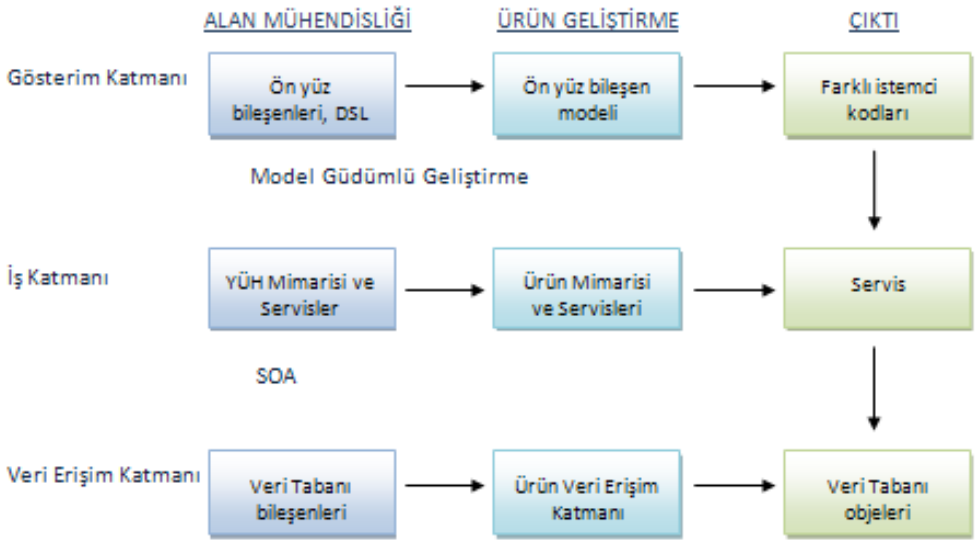
Tez çalışmasının vaka analizi bölümünde anlatılacak olan YÜH örneğinde platform çoklayıcı model sunulmuştur. SOA ile SPLE yaklaşımlarının birlikte kullanıldığı bu modelde YÜH kullanılarak gerçeklenen bir yazılım sisteminin farklı istemcilerle birlikte çalışabilmesi için yeniden kullanılabilir yazılım mimarisi, servisler ile oluşturulmuştur. Yani farklı tiplerdeki istemciler yazılım ürününe ait seçilen servisleri çağırabileceklerdir.

SOA kullanılarak YÜH mimarisinin uygulama platformuna göre özelleştirilebilmesi sağlanmıştır. Burada "bir kere yaz her yerde kullan" mantığı, PLSE kapsamında seçilen

ve yapılandırılan bileşenlerin ve servislerin farklı uygulama tiplerinde de ekstra geliştirme yapmadan kullanılabilmesidir.

Bu modelde farklı platformlar için kullanılabilecek yapıyı sağlamak için YÜH mimarisinin birleştirildiği bir diğer yöntem de Model Güdümlü Yazılım Geliştirme [37] yöntemidir. Seçilen ve ürüne göre uyarlanan bileşenler modellenir ve oluşturulan bu model dosyasından farklı istemci tiplerine göre otomatik olarak kod üretilerek yazılım sisteminin versiyonları elde edilir.

Şekil 4.1'de iç katmanlı yazılım sistemleri için sunulan model içinde bu yaklaşımların kullanımı gösterilmiştir.



Şekil 4.1 Platform çoklayıcı yazılım sistemi için SOA ve YÜH kullanım modeli

4.2.1 Sunum Katmanı

Sunum katmanı yazılım sisteminin kullanıcı ara yüzlerinden oluşur. Çalışmada kullandığımız modelde ***Alan Mühendisliği*** kapsamında alanın kapsamına ve gereksinimlerine göre geliştirilen *ön yüz bileşenleri*, daha sonra oluşturulacak ön yüz modelinden istemci tipine göre kaynak kodların otomatik olarak üretilmesini sağlayan *kod üreteçleri* geliştirilir. Burada geliştirilen ön yüz bileşenlerinin, YÜH kapsamındaki tüm ürünler göz önünde bulundurularak parametrik ve ayarlanabilir olması ve aynı

zamanda alanın gereksinim analizinde yer alan tüm fonksiyonları yerine getirmesi gerekmektedir(Login, alışveriş sepeti gibi).

Uygulama mühendisliği aşamasında kullanılacak olan bu bileşenlerin modellenebilmesi için bir *geliştirme veya modelleme aracı* olması yazılım üretkenliğini arttıracaktır. Bu geliştirme aracı ile yine Alan Mühendisliği kapsamında sağlanan bir *DSL* ile ön yüz bileşenlerinin birbirlerine bağlanması gerçekleştirilebilir. Oluşturulan model dosyasından istemcinin tipine göre kaynak kod üretecek kod üreteçleri ile gösterim katmanı için gerekli ana varlıkları sağlanmış olunur. Ürün geliştirme sırasında geliştirme aracı kullanılarak ön yüz bileşenleri seçilir ve ayarlanır. Gösterim katmanı çıktısı olarak istemci tipine göre kaynak kodlar elde edilir(Html, SWT gibi).

4.2.2 İş Katmanı

İş katmanında Alan Mühendisliği aşamasında mimarinin *orta katmanı* için gerekli bileşenler ve ürünlerde ortak *olarak kullanılan servisler* gerçeklenir. Bileşenlere örnek olarak ön bellek, Servlet, Session yönetimi, veri erişim bileşenleri, servislere örnek olarak kullanıcı yetkilendirme servisi, işlem yaratma servisi, işlem kayıt servisi verilebilir. Ürün geliştirme aşamasında ürün hattı mimarisi yapılandırılarak ürün mimarisi oluşturulur ve servisler seçilir. Eğer gerekiyorsa ürüne özgü servisler gerçeklenir. İş katmanı çıktısı olarak ana mimari ve ortak servisler elde edilir.

4.2.3 Veri Erişim Katmanı

Veri tabanı bileşenlerinin seçilmesi ve veri tabanı yönetimi bu aşamada yapılır. Geliştirilen YÜH, VTYS yazılımlarından (PostgreSQL, Oracle, MySQL, DerbyDB, vs) bağımsız olmalı ve hepsiyle uyumlu çalışabilmelidir. Web servis çağrıları sonucu elde edilen veri de bu katman ele alınmalıdır. Alan mühendisliği kapsamından veri tabanı bağlantı bilgisi, veri tabanı ismi, tablolar gibi bilgiler bir değişkenlik yöntemi kullanılarak parametrik hale getirilir. Ürün geliştirme sırasında bu parametreler ayarlanarak veri erişim katmanı ürünün ihtiyaçlarına göre oluşturulur. Bu katmanın çıktısı olarak veri tabanı ayarları ve veri tabanı nesneleri elde edilir.

Bu model ile birlikte PLSE'deki Uygulama Mühendisliği sürecinin sonucunda birden fazla istemci için çalışabilen yazılım sistemi elde edilmiş olunur. Yani, YÜH mimarisi istemci tipine göre özelleştirilebilmektedir.

Bir YÜH'nin işletim platformuna göre farklı ürünler oluşturabilmesi mevcut YÜH yaklaşımları tarafından da yapılabilmektedir [7]. Ancak istemci tipine göre farklı ürünler elde edebilmek için istemci tipi değişkeni seçildikten sonra her bir ürün için YÜH Uygulama Mühendisliği adımlarının uygulanması gerekmektedir. Bu da ürün gerçeklemesi sırasında aynı işlemlerin tekrar edilmesi anlamına gelmektedir.

Bu tekrarı önlemek ve yeniden kullanımı dolayısıyla üretkenliği daha da artırmak için istemci tipi değişkenliğinin ürün taşıma işlemi bittikten sonra da yapılandırılabilmesiyle mümkündür [35]. *Dinamik Yazılım Ürün Hatları* bu amaçla geliştirilmektedir.

3 katmanlı yazılım sistemlerinde bunu sağlamak için YÜH mimarisi, gösterim katmanında Model Güdümlü Yazılım Geliştirme ve orta katmanda da SOA ile birlikte çalışacak şekilde geliştirilmiştir. Böylelikle bu yazılım sistemlerinin farklı istemci tipleri için çalışan birer kopyası oluşturulabilmektedir.

Çalışmanın bir sonraki bölümünde bu model kullanılarak geliştirilen YÜH incelenecektir.

BÖLÜM 5

ADK UYGULAMALARI için YAZILIM ÜRÜN HATTI:
Örnek Uygulama Sistem Analizi

Günümüzde kurumlar, kullandığı yazılım sistemlerini kaliteli, dayanıklı, ölçeklenebilir, kolay üretilebilir, esnek, merkezi ve web tabanlı uygulamalar olarak hızlı bir şekilde geliştirmek istemektedirler. Kurumların rekabet ve müşteri memnuniyeti açısından en çok önem verdikleri hizmet sağlayıcıları Alternatif Dağıtım Kanalları (ADK) uygulamalarıdır. ADK uygulamaları, kurumsal firmalar tarafından sağlanan hizmetlerin farklı dağıtım kanalları üzerinden erişebilmesine yarayan uygulamalardır.

Hızlı, kaliteli ve düşük maliyetli ADK uygulamaları geliştirmek için sistematik olarak yeniden kullanıma göre alan mühendisliği çalışmaları yapan yazılım evi[1];

- Hızlı ve kolay geliştirme ve her ortamda çalışabilen yazılım varlıkları için bir ürün hattı mimarisinin geliştirilmesini,
- Bu uygulamalarda sıkça ihtiyaç duyulan ve kullanılmakta olan ekranlar, servisler için bileşenlerin geliştirilmesini,
- Sık değişen müşteri isteklerine hızlı bir şekilde cevap verebilmeyi amaçlamıştır.

Bu kapsamda yazılım üretkenliğini ve kalitesini artırmak için geliştirilen büyük ölçekli kurumsal ADK yazılım sistemlerinde kullanılmakta olan Graymound[2] yazılım ürün hattı önceki bölümde sunulan Platform Çoklayıcı Modeli gerçeklemek üzere YÜHM disiplini ile geliştirilmiştir.

1 OBSS Bilişim, www.obss.com.tr

2 Graymound, www.obss.com.tr/en/services/Graymound

Graymound; potansiyel müşterilerden alınan görüşler, ihtiyaçlar ve sektörü iyi bilen yazılım mimarları tarafından geliştirilmiş büyük ölçekli kurumsal yazılımların ortak özelliklerini barındıran ve ürünler arasındaki değişkenlikleri yönetebilen bir açık kaynak yazılım ürün hattı mimarisidir. Ortamdan bağımsız, istikrarlı ve hızlı geliştirim sağlaması açısından Java ile gerçeklenmiştir. Java; sunucu ve iş katmanlarını oluşturmak ve değişik veri kaynakları arasında veri alışverişini gerçekleştirmek açısından çeşitli seçenekler sunar.

Graymound da Java üzerine kurulduğundan Java'nın sunduğu esnek yapıyı sağlayabilir durumdadır. Bununla birlikte; Graymound, yazılım geliştiriciler için J2EE uygulamalarını hızlı bir biçimde geliştirme, kurumsal raporlama ve ortamdan bağımsız çözümler üretme konusunda güçlü özellikler içerir.

Platform Çoklayıcı modele göre gerçeklenmiştir. İstemci tipinin dinamik olarak yapılandırılabildiği bir YÜH yaklaşımı ile Platform Çoklayıcı model gerçeklenmiştir. Bileşenleri ve özellikleri tüm ürünlerden kullanılabilir yapabilmek için Servis Tabanlı Mimariden faydalanılmıştır. Bu konu bir sonraki bölümde detaylı olarak anlatılacaktır.

Örnek çalışmanın geri kalanında geliştirilen ürün hattı mimarisinde yeniden kullanım süreçlerinin nasıl gerçekleştirildiği, Alan Mühendisliği ve Uygulama Mühendisliği aktiviteleri, değişkenliklerin ve ortak noktaların nasıl yönetildiği anlatılacaktır. Bu ürün hattı mimarisi kullanılarak ürün geliştirme süreci örnek bir uygulama ile gösterilecektir. Son olarak bu ürün hattı yaklaşımın organizasyona sağladığı faydalar üzerinde durulacaktır.

5.1 Alan Mühendisliği

Yazılım Ürün Hatlarında yeniden kullanılacak bileşenler, yazılım varlıkları ve araç desteği Alan Mühendisliği aşamasında gerçeklenir. Graymound mimarisi içerisindeki Alan Mühendisliği aktivitelerinin amacı kolay, hızlı, standartlara sahip ve kaliteli bir yazılım üretim mekanizması sağlamaktır.

Yazılım Ürün Hatlarında alan mühendisliği kapsamında geleneksel yazılım geliştirme yöntemindeki gereksinim analizi, tasarım, geliştirme ve test süreçleri uygulanır. Bu süreçler uygulanırken geleneksel yöntemden farklı olarak tüm ürünler düşünülerek

yapılmalıdır. Alan mühendisliği süresince bu süreçler için çalışan ayrılması sistematik yeniden kullanım için daha uygun olmaktadır.

5.1.1 Alan Analizi

Alan analizi aşamasında, alanın tanımı yapılır. Tanıma göre alan kapsamı belirlenir.

5.1.1.1 Alan Tanımı

Büyük ölçekli kurumsal ADK uygulamaları genellikle veri kaydetme, güncelleme ve sorgulama işlemlerini içerir. İçerisinde çok fazla algoritma barındırmazlar. Bu uygulamalarda önemli olan verinin doğru, hızlı ve güvenli bir şekilde gönderilmesidir. Uygulamanın her zaman aynı şekilde davranması, devamlı çalışabilir durumda olması gereklidir. Bu uygulamalarda farklı istemcilerden gelen istekler için istemciye göre sonuçlar gönderilmesi gerekir.

Bu uygulamaların geçerli olduğu alanın kapsamındaki ürünler, sadece özellikler seçilerek ürün oluşturulan YÜHler düşünüldüğünde genelde karşılaşılan yapıdan biraz daha karmaşıktır. Çünkü her istemci tipine göre değişen ön yüz bileşenlerinin ve iş katmanı özelliklerinin belirlenmesi her zaman mümkün olmayabilir.

Graymound kapsamında bu bahsedilen yazılım ürünleri dışında bir ürün geliştirmek tavsiye edilmemektedir. Çünkü Ürün Hattının kapsamının dışında bir ürün geliştirmek istenirse ürüne özgü kod sayısı artacağından geliştirme süresi daha da uzayabilir.

ADK uygulamalarında orta katmanda geliştirilen bir yazılım çözümünün farklı kullanıcı ara yüzleri ile yeniden kullanılabilmesi istenmektedir. Yani; bir uygulamanın sunduğu bir servis şirket içindeki ağda yer alan bir bilgisayardan masaüstü istemcisi ile çalışabildiği gibi uzaktaki bir istemciden web tarayıcısı veya web servisleri ile de çağrılabilmelidir. YÜHM de bu yapıya uygun bir yeniden kullanım yaklaşımıdır.

ADK yazılımları; kurum içi ağ ile masaüstü, internet üzerinden web, SOAP1 protokolü ile web servis ve raporlama aracı ile rapor uygulaması olarak kullanılmak istenmektedirler. Örnek olarak bir banka, fatura ödeme işlem ekranını internet

[1] SOAP (Simple Object Access Protocol), Servis Tabanlı Mimaride servisler arası haberleşme için bir ara yüz sunan standartlaşmış bir protokolüdür.

bankacılığı, banka şubesi, bayi, çağrı merkezi gibi farklı dağıtım kanalları ile kullanmaktadır. Çoğu ADK yazılım sistemi her bir uygulama için ayrı yazılımları baştan geliştirmektedir. Graymound'da ise bu aşamada YÜHM yaklaşımı kullanılmaktadır.

5.1.1.2 Alan Kapsamı

Çalışmada sunulan ürün hattı mimarisi kapsamında veri tanımlama, girme, sorgulama işlemlerinde oluşan farklı istemcilerden gelen isteklere uygun sonuçları dönen bir ön yüz ve orta katman sunulmaktadır. Burada kapsamı belirleyen en önemli özelliklerden biri dinamik olarak yapılandırılabilen istemci türleridir.

Graymound mimarisinde alan gereksinimlerine göre dinamik olarak yapılandırılabilen istemci tipleri şunlardır:

- Masaüstü Uygulamaları
- Web Uygulamaları
- Web Servis Uygulamaları
- Rapor Uygulamaları

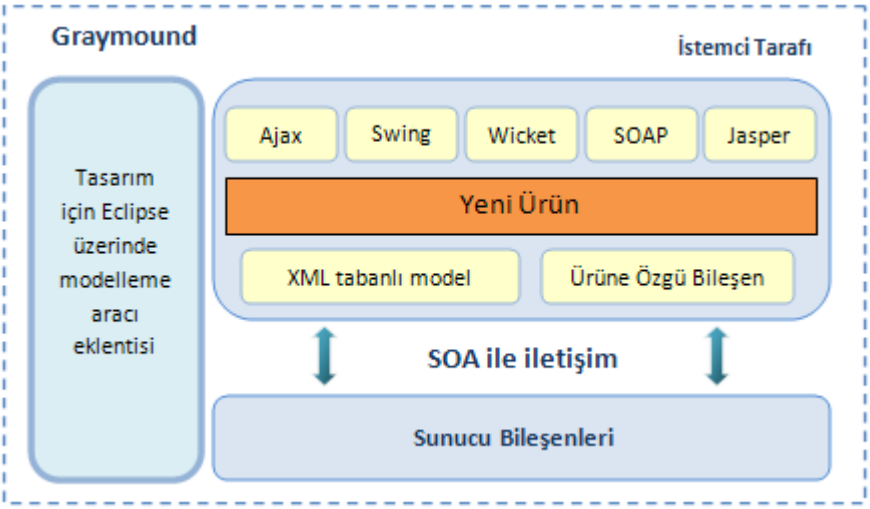
Graymound ile yapılan ürün geliştirmede gerçeklenen model ve servis ile bu 4 tip uygulamadan hepsi elde edilebilir. Yani model ve servis bir kere geliştirilir ve 4 farklı uygulama için de kullanılabilir. Yazılan bir model hem Swing hem de Ajax ön yüz içeriklerine çevrilebilir. Aynı zamanda yazılan bir servis bu dört uygulamadan da çağrılabilmektedir.

Gerçeklenen yazılım sistemleri J2EE tabanlı uygulamalar olduğu için J2EE standartlarına göre geliştirilmiş bir uygulama sunucusuna ihtiyaç duymaktadırlar.

5.1.2 Ana Varlıklar

Alan Mühendisliği kapsamında birlikte çalışılan müşterilerin ortak ihtiyaçlarına göre istemci ve sunucu tarafında ortak yazılım varlıkları geliştirilmiştir. Her üründe ayrı yapılandırmalarla bu varlıklar kullanılabilmektedir.

Şekil 5.1'de Graymound mimarisinin genel bileşenleri ve yapısı yer almaktadır. Eclipse[1] tabanlı bir geliştirme ortamı ile istemci ve sunucu tarafındaki bileşenler ve servisler yapılandırılabilir hale getirilmiş ve ürün geliştirme sırasında yeniden kullanımı sağlanmıştır.



Şekil 5.1 Graymound mimarisi

YÜH kapsamındaki **sunum katmanı** bileşenler GMClient, GMQooxdoo ve GMCommon kütüphaneleri içerisinde, **iş katmanı** bileşenler GMServer ve GMAdministration kütüphaneleri içerisinde, **veri erişim katmanı** da GMServer kütüphanesi içerisinde gerçeklenmiştir.

Graymound içerisinde, J2EE tabanlı ADK uygulamaları gerçeklendiği için J2EE standartlarına uygun bir uygulama sunucusu olan JBoss[2] ve VTYS yazılımı için açık kaynaklı ve taşınabilir bir sistem olması dolayısıyla DerbyDB[3] bulunur.

Alan mühendisliği sürecinin çıktısı olarak geliştirilen yazılım elemanları ve bileşenler;

- Tasarım ve Geliştirme ana varlıkları,
- Gereksinim Analizi ana varlıkları,

[1] Eclipse: Java, Pyhton, Javascript, C/C++ gibi programlama dilleri için eklenti tabanlı bir yazılım geliştirme aracı. www.eclipse.com

[2] JBoss: J2EE tabanlı, ücretsiz ve açık kaynaklı bir uygulama sunucusu. Graymound ile birlikte 5.1.0 versiyonu yüklü olarak gelmektedir. Detaylı bilgi için www.jboss.org

[3] DerbyDB: Apache tarafından geliştirilen ücretsiz, açık kaynaklı ve ilişkisel bir VTYS yazılımı. Graymound ile birlikte Derby Eclipse eklentisi yüklü olarak gelmektedir. Detaylı bilgi için db.apache.org/derby

- Test ana varlıkları olarak ele alınmıştır.

Tasarım ve geliştirme aşamasında yeniden kullanılacak ana varlıklar istemci tarafındaki ana varlıklar ve sunucu tarafındaki ana varlıklar olarak ayrılmaktadır.

5.1.2.1 Sunum Katmanı için İstemci Tarafındaki Ana Varlıklar

Graymound, zengin geliştirme aracı desteği sağlar; kullanıcı ara yüzleri hazırlamak için bir editöre sahiptir. Yazılımcılar bu editör ile zengin ara yüzlü web uygulamaları geliştirip dağıtımını yapabilir. Tüm Graymound mimarisi J2EE açık kaynak standartları üzerine kurulmuştur. Sunum katmanı Swing, SWT, Ajax, Wicket, GWT, JasperReports ve SOAP teknolojileri üzerine inşa edilmiştir.

Alan mühendisliği kapsamında istemci tarafında ana varlık olarak tüm ürünler tarafından ortak kullanılan;

- Ön yüz bileşenleri,
- Farklı istemci tipleri için içerik sağlayıcı,
- Ön yüz modelinden otomatik kaynak kod oluşturan kod üreteci sunulmaktadır.

Graymound Ürün Hattı mimarisi geliştiricilerin daha kolay ve hızlı geliştirme yapabilmelerini sağlamak için bir alana özgü modelleme aracı ile birlikte DSL kullanmaktadır. Sağlanan DSL alan mühendisliği aşamasında geliştirilen ana varlıklardan biridir. Geliştirme aracı eklentisinin sağladığı ürün hattı mimarisine özel Eclipse görünümüne (Eclipse Perspective) geçerek ürün hattının sağladığı bütün özellikler görülebilmektedir. Graymound bu ana varlıklar için değişkenlik yönetimini yapılandırma dosyaları kullanarak gerçekleştirmiştir.

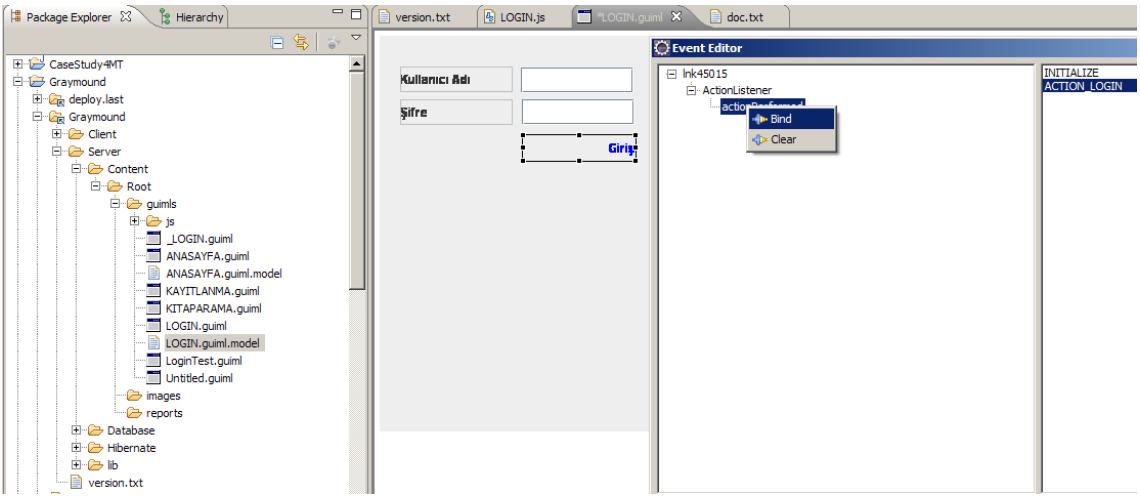
Geliştirme Aracı: Ön yüz bileşenlerinin seçilmesi ve yapılandırılması için Eclipse üzerine yazılan bir eklenti ile modelleme aracı sağlanmaktadır. EKA-1'de gösterilen yapıda istemci tarafında değişkenliğin seçilmesi için bu modelleme aracı kullanılmaktadır.

Kullanıcı Ön yüz Modelleme: Ön yüzler için ürüne özgü bileşenlerin seçilerek modellenmesi sonucu GUIML adı verilen XML[1] tabanlı bir model dosyası oluşmaktadır.

[1] XML: Extensible Markup Language http://www.w3.org/XML/

Bileşen ve özellik seçme işlemi için Eclipse üzerine yazılan bir eklenti ile geliştirme ortamı üzerinde bir araç sağlanmıştır. Oluşturulan bu model dosyasında seçilen bileşenin ve özelliğin ayarları ve birbirleri arasındaki bağlantı yapılmış durumdadır. Şekil 5.2'de geliştirme aracı ile oluşturulan örnek bir model dosyası görülmektedir.

Model Dönüştürme: Ürün Hattı, çalışma anında bu modeli koda dönüştürerek istemcinin istediği tipte seçilen İçerik Sağlayıcı özelliğine göre kod üreterek sonuç döner.



Şekil 5.2 Örnek GUIML modeli

Sunum katmanı bileşenler ve özellikler, değişkenlik noktaları, değişkenler ve açıklamaları ile birlikte Çizelge 5.1'de gösterilmektedir.

Çizelge 5.1 Sunum katmanı özellikleri

ANA VARLIK	DEĞİŞKENLİK NOKTASI	AÇIKLAMA	DEĞİŞKENLER	DEĞİŞKEN BAĞLAMA ZAMANI
Tarayıcı (GMBrowser.xml)	Boyut	Ana ekranın boyutu	1024,768 500,700 550,850	Kurulum zamanı
	Görünüm	Masaüstü uygulamaları için ekran görünümü	GMLookandFeel DefaultLookandFeel	Kurulum zamanı
	Giriş Sayfası	Kullanıcı giriş sayfası	LOGIN.guiml *.guiml	Başlatma zamanı

	Resize	Ekranın kullanıcı tarafından yeniden boyutlandırılabilmesi	true false	Başlatma zamanı
İçerik Sağlayıcı	İstemci Tipi	İstemcinin tipine göre içerik sağlayıcı seçerek farklı ürünler oluşturmayı sağlar.	GMJavaContext GMAjaxContext GMSOAPContext GMReportContext	Çalışma zamanı
Araç Çubuğu	Fonksiyon listesi	Üst menüde yer alacak butonlar, ikon resimleri	NEW SAVE DELETE SEARCH CLEAR	Başlatma zamanı
Yerleşim	Layout	Ön yüzün yerleşme düzeni	XYLayout FlowLayout GridLayout	Başlatma zamanı
Kullanıcı Girişi	Sanal Klavye Dili	Kullanıcı giriş sanal klavye ayarları yapılır.	Tr En De …	Çalışma zamanı
	İçerik tipi	Kullanıcı adı ve şifre alanının karakteri kısıtlaması	Plain Alpha Numeric Alphanumeric	Çalışma zamanı
	Klavye Tipi	Şifre girilecek alanda açılan sanal klavye tipi	Q F	Çalışma zamanı
Veri Arama ve Görüntüleme	Validate from list	Aranan verinin listede olup olmadığını gösterme	true false	Çalışma zamanı
	Sorgu harf formatı	Sorgu parametrelerinin harf duyarlılığı	Lowercase Uppercase Nocase	Çalışma zamanı
	Sorgu dosyası	LOV adı verilen sorgu dosyaları	LOV*.xml dosyaları ile	Çalışma zamanı
Tablo	Kayıt Seçme	Tablodaki satırların nasıl seçileceğini belirtir.	Single Single Interval Multiple Interval	Çalışma zamanı

			None	
	Sıralama	Tabloda kolona göre sıralama yapabilme	true false	Çalışma zamanı
	Otomatik Boyut	Tabloda hücrelerin otomatik boyutlandırılması	ON OFF	Çalışma zamanı
Veri Listeleme	Veri Kaynağı	Veri kaynağı seçilir.	*datasource.xml ile	Çalışma zamanı
	Veri Tablosu	VT Tablo Adı seçilir.	Modelleme ile	Çalışma zamanı

Bunların dışında her ürün için bulunması gereken ana varlıklar aşağıda listelenmiştir.

- *GMBeans.xml* dosyasında modelleme sırasında kullanılabilecek bileşenler seçilmektedir. Burada istenirse ürüne özgü geliştirilen ön yüz bileşeni de girilebilir.
- **BeanInfo.xml* dosyalarında ön yüz bileşenlerine ait fonksiyon, olay ve özelliklerden hangilerinin modelleme sırasında kullanılabileceği seçilmektedir.
- *GMPalette.xml* geliştirme ortamında modelleme aracında yer alan bileşen paletinde hangi bileşenlerin hangi grupta yer alacağı seçilir.
- *GMListener.xml* dosyasında ön yüzde kullanıcının girdiği bir olayı dinleyecek ve gerekli işlemleri yapacak Java sınıfları seçilmektedir.

5.1.2.2 Sunucu Tarafındaki Ana Varlıklar ve Değişkenlikler

Alan mühendisliği kapsamında geliştirilen sunucu tarafındaki ana varlıklar oluşturulacak yazılım ürünlerinin iş katmanı ve veri tabanı erişim katmanındaki bileşenlerini ve servisleri içerir. Bu ana varlıklar YÜH kapsamındaki her ürün seçilebilecek şekilde esnektir.

Sunum katmanıyla bağlantılı olarak İçerik Sağlayıcı için geliştirilen **içerik sağlayıcılar** da bunlardan biridir.

Alana Özgü Modelleme: İstemci tipine göre uygulamayı farklılaştırmayı sağlarken servislerin ortak kullanılmasını sağlayan yapı buradan gelmektedir. Burada Servis

Tabanlı Mimarinin ortaklıkların ve değişkenliklerin yönetimi için kullanılmaktadır. Servislerin seçilme ve çağrılma işi geliştiriciler tarafından model içerisinden yapılır. Eclipse üzerinde sağlanan editör ile model üzerinde servislerin seçilmesi ve parametrelerinin girilmesi kolayca yapılabilmektedir. Graymound'un kullanıcı ara yüzü sayfaları (GUIML) çözümlenerek JavaScript, Html, Java kodlarına çevrilirler. Bunu sağlayan Graymound mimarisi içersinde geliştirilen ana varlıklardan biri olan Alana Özgü Dildir.

Servis Tabanlı Mimari: Modelde seçilen değişkenler ile servisin çağrılması ön yüzden yapılmaktadır. Ana varlık olarak sunulan servislerde değişkenlik yönetimi servis parametreleri ile sağlanır.

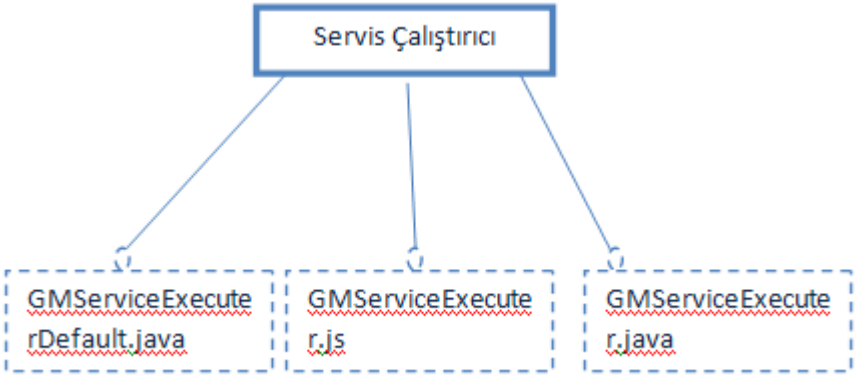
İçerik Sağlayıcı: Gelen isteğin tipini anlayarak uygun formatta cevap döner. İsteğin tipi gelen *GMRequest* objesindeki parametre ile anlaşılmaktadır. Bu sayede çalışma anında dinamik yapılandırma ile **Platform Çoklayıcı** model gerçeklenmiş olur. Bir yazılım sisteminin hangi istemci tipleri ile çalışacağı özelliği için yapılandırma dosyaları kullanılarak değişkenlik yönetimi sağlanır. İçerik sağlayıcı bileşeni devam eden gerçekleme tamamlandığında Çizelge 5.2'deki özellikleri sağlayacaktır.

Çizelge 5.2 İçerik sağlayıcı özellikleri

ANA VARLIK	DEĞİŞKENLİK NOKTASI	AÇIKLAMA	DEĞİŞKENLER	DEĞİŞKEN BAĞLAMA ZAMANI
İçerik Sağlayıcı	Masaüstü sunum katmanı	Masaüstü uygulamaları için hangi teknolojinin kullanılacağı seçilir.	Swing SWT	Geliştirme zamanı
	Web sunum katmanı	Web uygulamaları için hangi teknolojinin kullanılacağı seçilir.	Qooxdoo Wicket GWT	Geliştirme zamanı
	Web servis	Web servis teknolojisi	SOAP SOAP2	Geliştirme zamanı
	Raporlama	Rapor teknolojisi	JasperReportPDF JasperReportHTML	Geliştirme zamanı

Bu içerik sağlayıcılarına ürüne özgü yeni içerik sağlayıcılar eklemek için *GMContext* soyut sınıfı gerçeklenmelidir.

İş katmanındaki ana varlıklardan servis çalıştırıcı için ortaklıklar ve değişkenliklere ait özellik ağacı Şekil 5.3'de gösterilmektedir. Burada servis çalıştırıcı bileşeni için 3 tane değişken sunulmuştur. Bağlantı noktalarındaki işaret bu özelliklerden herhangi birinin seçimli olarak seçilebileceğini gösterir [15].



Şekil 5.3 Servis çalıştırıcı bileşeni özellik ağacı

İş Katmanındaki diğer ana varlıklar aşağıda belirtilmiştir.

Servis Sınıfları: Servis yükleyici için 2 tane değişken bulunmaktadır. Bunlardan birincisi varsayılan değer olan *DBLoader*, servis sınıflarını veri tabanından alır. İkincisi servis sınıflarını XML dosyasından alır.

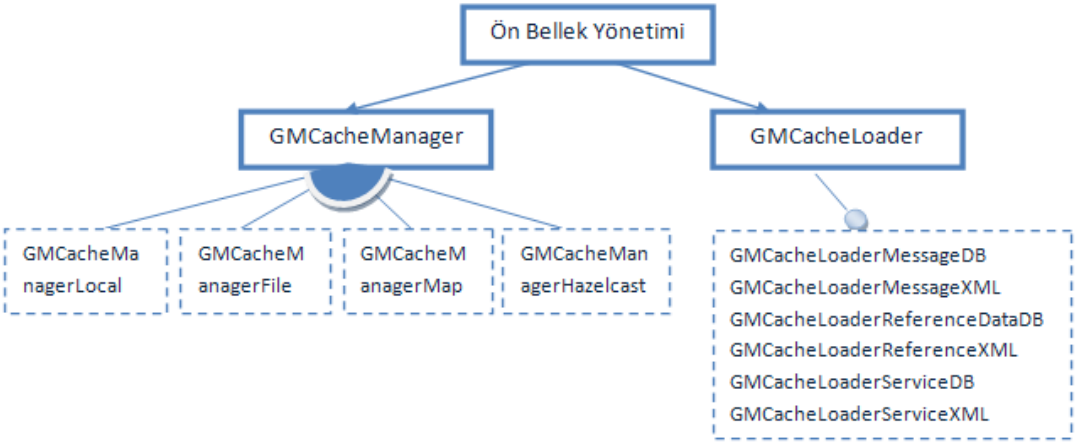
Doğrulama ve Yetkilendirme: GMConnection. xml yapılandırma dosyası ile belirlenir.

EJB Uzak Çağrıları; properties dosyası ile seçilir.

Mesaj verisi; *GMMessageConfiguration* dosyasından okunur. Bu değişim noktasında veri tabanı yükleyici ve XML yükleyici olmak üzere 2 tane değişken vardır.

Referans verisi; *GMReferenceDataConfiguration* dosyasından okunur. Bu değişim noktasında veri tabanı yükleyici ve XML yükleyici olmak üzere 2 tane değişken vardır.

Önbellek Yönetimi; önbellek objelerinin yönetiminin nasıl yapılacağı, hangi objelerin önbelleğe alınacağı, *CacheManager* ve *CacheLoader* değişim noktaları seçilerek belirlenir. Şekil 5.4'de önbellek bileşeni için değişim noktaları ve değişkenleri gösteren özellik ağacı yer almaktadır.



Şekil 5.4 Önbellek yönetim bileşeni özellik ağacında değişkenlik gösterimi

Önbellek ana varlıklarından *GMCacheManager* değişim noktası seçimli, yani birden fazla seçilebilen özellik iken, *GMCacheLoader* alternatif yani en fazla bir tane seçilebilen özelliktir [15].

Sunucu özellikleri; mimarinin sunucu tarafının özellikleri, sunucu başlatma parametreleri, ürün mimarisinin sağlayacağı içerik (uygulama) tipleri *GMServerConfiguration* dosyasında belirlenir.

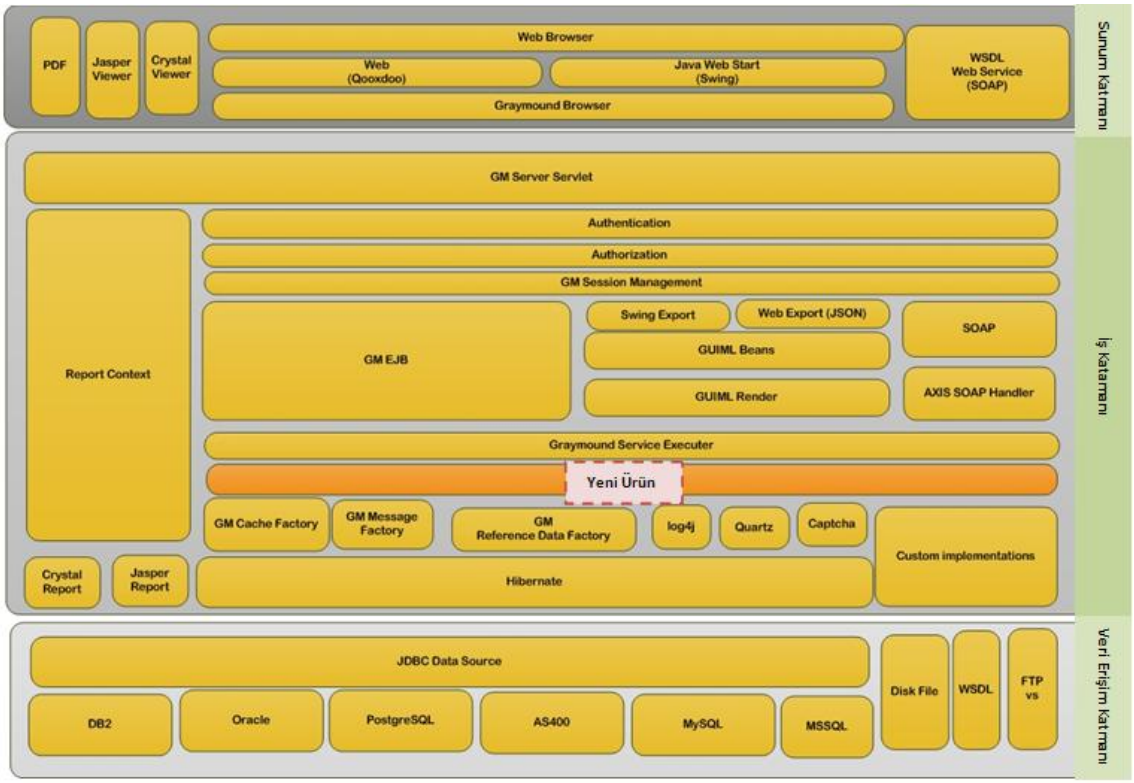
Dil Seçenekleri; oluşturulacak sayfalar için modelleme sırasında seçilebilecek dil seçeneklerinin seçilmesi (tr, en, de, …)

VTYS: Graymound'da, veritabanı katmanıyla JDBC veya Hibernate aracılığıyla iletişim kurulur. Böylece uygulamalar veritabanı türünden bağımsız duruma getirilip, uygulama geliştirici, gereksinimine göre uygun JDBC sürücüsünü yüklemek koşulu ile PostgreSQL, MySQL, Oracle, MS-SQL, vs VTYS yazılımı kullanma konusunda özgür konuma getirilir. Graymound kurulumu ile birlikte Derby VTYS yazılımı yüklü gelmektedir.

Veri Kaynağı; Graymound-ds. xml dosyasında belirlenir. Varsayılan değer olarak kurulum ile birlikte gelen ADK veri tabanı elemanları (tablo, fonksiyon, yordam, vb) kullanılmaktadır.

Şekil 5.5'de YÜH mimarisinde sunum katmanı, iş katmanı ve veri tabanı erişim katmanındaki bileşenler gösterilmektedir. YÜH mimarisinin sağladığı esnek ve yeniden kullanılabilir yapı içinde yeni bir ürün ekleme işleminin mimarinin neresinde olduğu

yeni ürün ibaresi ile gösterilmiştir. Buna göre yeni ürünün mimarisi YÜH mimarisini referans olarak almaktadır.



Şekil 5.5 Graymound bileşenleri

Kod Derleme Otomasyonu ve Projelerin Bağımlılık Yönetimi: Kodu derleyecek uygulama sisteminin paketini alan ve onları istemci tiplerine göre *WAR* veya *JAR* dosyaları halinde yayımlayan taşıma betikleri bulunmaktadır. Derleme aracı olarak ANT[1] veya Maven[2] kullanılmaktadır. EKA-2'de bir Graymound projesi örneği için kaynak kod derleme dosyası bulunmaktadır.

Ortak Kullanılan Parametrik Servisler: Alan mühendisliği kapsamında tüm ADK uygulamalarında ortak kullanılan servisler geliştirilmiştir. Servis kaynak kodları ADCCore Kütüphanesi içerisindedir. Çizelge 5.3'de tüm ADK ürünleri için ortak ADK servisleri bulunmaktadır.

[1] ANT: Apache tarafından sunulan proje yönetim ve derleme aracı.

[2] Maven: Apache tarafından geliştirilen yazılım projeleri derleme, taşıma ve yönteme aracı.

Çizelge 5.3 Ortak kullanılan servisler ve parametreleri

SERVİS ADI	AÇIKLAMA	GİRDİ PARAMETRELERİ
ADC_CORE_CREATE_USER	Yeni Kullanıcı Oluşturma	Name username email password re_password
ADC_CORE_AUTHENTICATE	Kullanıcı Denetimi	username password language channel sso_token
ADC_CORE_AUTHORIZE	Yetkilendirme	username language channel
ADC_CORE_PROCESS_EXECUTE	İşlem Çalıştırma, yeni işlem kaydı	process_code process_key pass_execution amount amount_code
ADC_CORE_PROCESS_APPROVE	İşlem onaylama	process_id pass_execution
ADC_CORE_PROCESS_LOG	İşlem bilgilerini kayıtlama	process_code process_key pass_execution amount amount_code user_id process_date

ADC_CORE_PROCESS_LIST	İşlemleri listeleme	process_code

ADK servisleri için değişkenlik yönetimi servislerin parametreleri ile sağlanmaktadır. Ortak servisler tüm ADK yazılım sistemleri tarafından çağrılabilecek şekilde parametrik yapıdadır.

Graymound kurulumu ile birlikte ana varlıklar için kurulum ve geliştirme zamanında bağlanan değişim noktalarına varsayılan değişkenler seçili olarak gelmektedir. EKA-3'de ana varlıklar için ayarlanmış varsayılan değerleri içeren yapılandırma dosyaları bulunmaktadır.

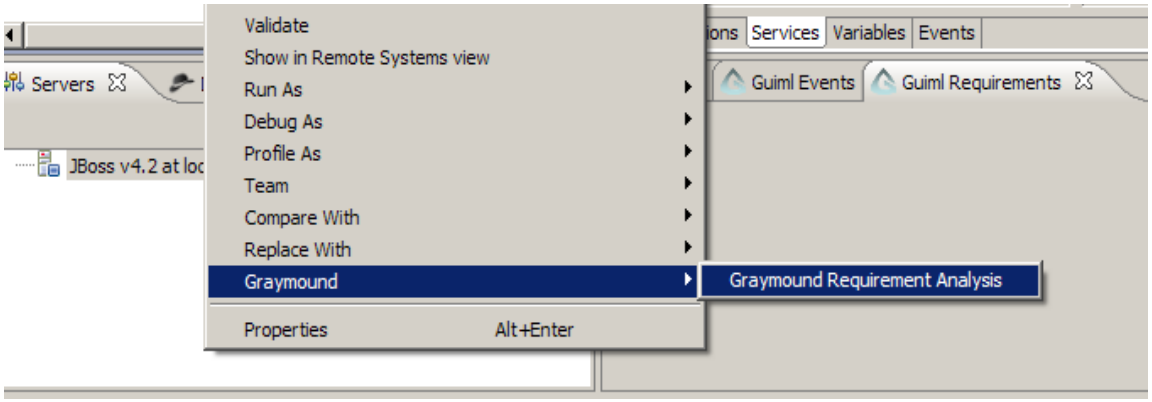
5.1.2.3 Gereksinim Analizi Ana Varlıkları

Yazılım ürün hattı mimarisi sağladığı gereksinim analizi aracı ile bu süreç için de yeniden kullanılabilir yazılım varlıkları sunmaktadır. Yeniden kullanım, alan mühendisliğinde ürün hattı mimarisinin bileşenleri için yazılan gereksinim analizi dokümanının ürünlerde de kullanılması ve sistem analistleri tarafından hazırlanan ön yüz modelinden elde edilen çıktıyla ürünün tüm sayfalarını içeren genel bir gereksinim analizi raporu çıkarılabilmektedir.

Doküman Üreteçleri: Alan mühendisliğinde gereksinim analizi süreci için ürün gerçekleme sırasında girilen analiz modeline göre dokümantasyonun otomatik oluşturulmasını sağlayan bir üreteç sağlanmıştır. Analiz dokümanı oluşturulurken kullanılan model gerçekleme sırasında da kullanılacak olduğundan gerçekleme sırasında hata çıkma oranı çok düşmektedir. Şekil 5.6'da bir Eclipse eklentisi olan Graymound gereksinim analizi aracının kullanımı gösterilmektedir.

Seçilen model dosyasına sağ tıklayıp Graymound →Graymound Requirement Analysis sekmesi seçilerek analiz dokümanı oluşturulur.

Model güdümlü geliştirme yönteminin analiz sürecinde de kullanılmasıyla sağlanan bu yapı sayesinde hem analiz dokümanı yazma iş gücünden hem hata çözme iş gücünden kazanç söz konusu olmaktadır.



Şekil 5.6 Gereksinim analizi doküman üreteci

5.1.2.4 Ürün Testi için Ana Varlıklar

Yazılım testi, yazılım geliştirme süresince zaman alan ve önemli bir iş gücü gerektiren bir süreçtir. Bir yazılım ürün hattının, kod seviyesinde yeniden kullanımının yanı sıra yazılım geliştirme süreçlerinden biri olan yazılım testi için de yeniden kullanılabilir yapılar sunması gerekmektedir. Bir YÜH organizasyonun başarısı iyi yönetilmiş ve otomatikleştirilmiş bir test sürecine de bağlıdır.

Graymound, sağladığı Eclipse tabanlı geliştirme ortamı ile yeniden kullanılabilir test elemanları içermektedir. Bunlar daha önce bahsedilen;

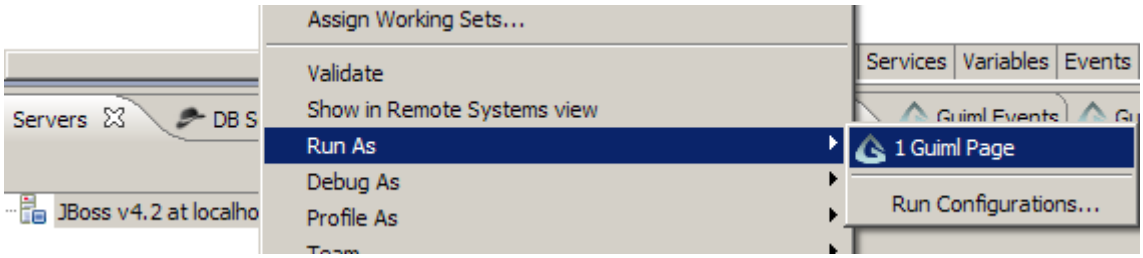
kullanıcı giriş servisi, kullanıcı yetkilendirme servisi ve işlem tanımlama servisinin testlerini içermektedir.

Bu servisleri test kipinde çağırdıktan sonra ürünü test eden kişi ürünün diğer servislerinin doğru çalışıp çalışmadığını yapmak istediği test durumları girdileri ile ürünün bir test çıktısını elde edebilmektedir.

Graymound ile geliştirilen bir model dosyasının web uygulaması olarak çalışmasının testi için yapılan isteğe test parametresi eklenir.

http://localhost/GMAjax/index.html?**test=true**&firstPage=LOGIN.js&loginPage=LOGIN.guiml&language=tr

Masaüstü uygulamasının testi için test edilecek sistemin kullanıcı yetkilendirme sayfası ayarlanmalıdır. Şekil 5.7'de görüldüğü gibi bu işlem sonunda model dosyasına sağ tıklayıp "Run as → Guiml Page" seçildiğinde sayfa test kipinde açılır. Test durumları için ayarlanması gereken bağlantı, yetkilendirme gibi işlemler YÜH tarafından yapılır.



Şekil 5.7 Ön yüz ve servislerin test durumlarının hazırlanması

5.1.3 Ürüne Özgü Geliştirme

Ürün geliştirme sırasında Graymound ürün hattı mimarisinin karşılamadığı bir özelliğe gereksinim duyuluyorsa bunun için Graymound'un genişletme noktalarından yararlanılabilir. Ürün mimarisine harici bileşenler, .jar dosyaları ve buna bağlı istemci tarafında XML yapılandırma dosyaları eklenebilir.

İstemci tarafında *lib/ext* klasörü altına ürüne özgü bileşenlerin kaynak kod paketleri konularak bileşen ekleme, değiştirme yapılabilir.

Yine sunucu tarafında lib/ext klasörü altına ürüne özgü bileşenlerin ve servis sınıflarının kaynak kod paketleri konularak bileşen ekleme, değiştirme yapılabilir.

Ürüne Özgü Servisler: Eğer YÜH tarafından sağlanan ADK servisleri gerçeklenecek yeni ürünün ihtiyaçlarını karşılamıyorsa değiştirilmek istenen veya yeni eklenmek istenen servisler EKB-1'deki kod örneği gibi oluşturulabilir.

5.2 Değişkenlik Yönetimi

Bir ürün hattı mimarisinin, farklı ürünlerde değişkenlik gösteren bileşenleri ve yazılım varlıklarını sağlaması ve yönetmesi gerekmektedir. Graymound bunu yaparken aşağıdaki belirtilen değişkenlik mekanizmalarını kullanmıştır.

- Yapılandırma dosyaları
    - GMServerConfiguration.xml
    - GMServiceConfiguration.xml
    - GMBeans.xml, GmBrowserConfiguration.xml gibi
- Eclipse tabanlı bir geliştirme aracı ile istemci tarafındaki bileşenlerin kullanılması

- Model güdümlü programlamaya göre geliştirilen model dosyası
- Yansıma (Reflection)
    - Seçilen bileşenlerin değişkenliklerinin ayarlanması ve
    - Servislerin çalıştırılması için kullanılır.
- Dinamik Sınıf Yükleme
    - Yapılandırma dosyasında seçilen sunucu tarafındaki değişkenlerin seçilmesi için kullanılır. Örnek:

```
<property name="transaction.coordinator"
value="com.graymound.server.coordinator.GMTransactionCoord
inatorDefault"/>
```

- Kalıtım
- SOA ile farklı platformlardan çalışabilme; kod tarafındaki platforma göre içerik sağlayıcı seçilmesi GMContext sınıfının gerçeklemeleri ile sağlanır.

5.3 Organizasyon

Örnek çalışmalarda kullanılan YÜH, ağrılıklı olarak Türkiye'de yazılım çözümleri sunan, iki ayrı yerleşkede (İstanbul ve Gebze) 60'ın üzerinde çalışanı bulunan bir yazılım evi olan OBSS Bilişim bünyesinde gerçekleştirilmiştir. Organizasyon, bir yazılım ürün hattı olarak geliştirdiği sistemi, sunduğu yazılım çözümlerinde kullanmaktadır. Her müşteri için Ürün Hattı Mimarisinin ortak bileşenleri ve servisleri özelleştirilerek müşteriye özgü projeler geliştirilebilmektedir.

Graymound YÜH, yazılım mimarlarının mevcut proje tecrübeleri ve müşterilerden gelen ihtiyaçlar ile şekillenmiştir. Mevcutta bankacılık, finans, sigorta, hastane, muhasebe gibi birçok ADK uygulamaları yazılımı Graymound YÜH kullanılarak geliştirilmiş ve geliştirilmeye devam etmektedir. Graymound ekibi, müşterilerden gelen geri dönüş ve isteklere göre YÜH güncel tutmaktadır.

Organizasyon içerisinde Alan Mühendisliği aktiviteleri için kurumsal J2EE uygulamaları konusunda tecrübeli, sektörün ihtiyaçlarını iyi bilen ve sadece bu iş için çalışan yazılım mimarları ve proje yöneticileri bulunmaktadır.

Uygulama mühendisliği için yazılım mühendisleri müşteri ile birlikte çalışarak istenen yazılım sistemlerini hızlı ve kolay bir şekilde gerçekleyebilmektedir. Bu süreçte çalışan personelin YÜH kullanımını ve mimarinin detaylarının öğrenmesi çok zaman almamaktadır. Böylelikle yazılım geliştirici kaynağı bulmak zor olmamaktadır. Geliştirilen ürünler, daha önceden kullanılmış ve test edilmiş bileşenlerden oluştuğu için yazılım kalitesi yüksek ve piyasaya sürme süresi düşüktür.

BÖLÜM 6

ÖRNEK SİSTEM TASARIMI ve GERÇEKLEMESİ

Yazılım ürün hattı başlığı altında anlatıldığı üzere, yeni bir ürün geliştirmek için uygulama mühendisliği aşamasında mimarinin sağladığı ana yazılım varlıkları, ürünün ihtiyacına göre belirlenir ve yine ürünün ihtiyacına göre bileşenlere ait özellikler seçilir. Buradan yola çıkarak bu bölümde, bir önceki bölümde bahsedilen Alan Mühendisliği kapsamında oluşturulan ve tüm ürünler tarafından kullanılabilen ana yazılım varlıklarının, belirlenen bir uygulama için kullanılarak yeni bir yazılım ürünü geliştirilmesi gösterilecektir.

Ürün geliştirme aktivitelerinin tümünü gösterebilmek için örnek çalışma temel işlemleri yerine getiren basit bir ADK uygulaması seçilmiştir. Mimarinin sağladığı bileşenlerden bazıları seçilerek yetkilendirme isteyen bir *işlem sayfası* hazırlanmıştır. Yetkilendirme için de yine YÜH ile *kullanıcı kayıt* ve *giriş sayfası* yapılmıştır. Bu uygulama ile kullanıcılar sisteme kayıt olabilmekte ve belirledikleri şifreler ile giriş yapabilmektedirler. Gerçeklenen yazılım sistemi seçilen farklı ADK uygulamaları olarak çalışabilecektir.

Sonraki bölümlerde ürün geliştirme sırasında, geliştirme ortamının kurulumundan ürün testine kadar olan süreçlerde seçilen değişkenlikler ve bu değişkenliklere bağlı mimari tarafından sunulan değişkenler gösterilecektir.

6.1 Ürün Hattı Mimarisinin Kurulumu

Ürün Hattı Mimarisinin kurulumu için ilk olarak geliştirme ortamı hazırlanır. Daha sonra mimarinin bir örneği oluşturulacak şekilde yeni proje oluşturulur.

6.1.1 Geliştirme Ortamının Hazırlanması

Kullandığımız yazılım ürün hattı, geliştirilecek olan yeni uygulamada sistem tasarımı ve gerçekleme sürecini hızlandırmak ve bileşenlerin seçilmesini sağlamak için Eclipse üzerinde bir geliştirme aracı sunmaktadır. Örnek Yazılım Ürün Hattı;

- Değişkenlerinin seçilmesi,
- Sunum katmanının modellenmesi,
- Ortak servislerin çağrılması,
- Ürüne özgü servislerin gerçeklenmesi ve çağırılması,
- Analiz varlıklarının seçilmesi,
- Test varlıklarının seçilmesi,
- Ürüne özgü bileşenlerin gerçeklenmesi

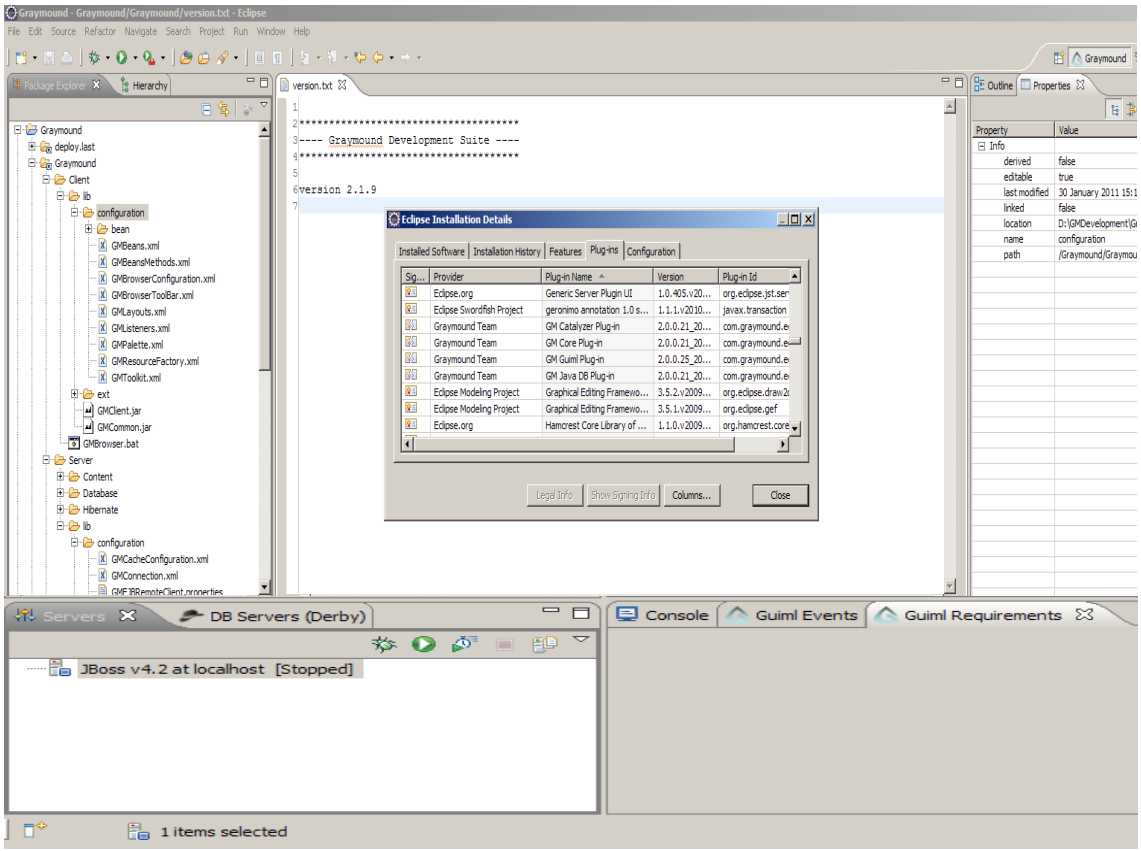
için alan mühendisliği kapsamında geliştirilen Eclipse eklentisi ile araç desteği sağlamaktadır.

Ürün hattı ile ürün geliştirebilmek için bu geliştirme aracı üzerinde Graymound eklentileri kurulmalı ve geliştirilecek ürüne göre ayarlamaları yapılmalıdır.

Bunların dışında; J2EE uygulamaları yapısı gereği gelen istekleri karşılayacağı bir uygulama sunucusuna ihtiyaç duymaktadır. Örnek uygulamamızda J2EE standartlarına göre geliştirilmiş gerçek uygulamalarda sıkça kullanılan JBoss uygulama sunucusu kullanılacaktır. Graymound geliştirme ortamın kurulumu ile birlikte yukarıda belirtilen Eclipse geliştirme aracı, Graymound Eclipse eklentisi, JBoss sunucusu 5.1.0 versiyonu yüklü olarak gelmektedir.

Şekil 6.1'de kurulum sonrasında çalıştırdığımız Graymound geliştirme ortamının ekran görüntüsü yer almaktadır. Sol tarafta Graymound projesi altında Graymound/Client ve Graymound/Server isimli klasörlerde ürün geliştirmek için gerekli olan istemci ve sunucu tarafındaki değişkenliklerden bazılarının yönetildiği yapılandırma dosyaları yer almaktadır. Örnek yazılım sisteminde, bütün bileşenlerinin özelliklerini değiştirmiyoruz ve E2'de belirtilen varsayılan değerleri kullanıyoruz.

Kurulum tamamlandıktan sonra ortamın sistem tasarım ve gerçeklemeye hazır hale getirilmesi için **değişken bağlama zamanı,** "kurulum" olan değişkenler seçilerek bir takım yapılandırmalar yapılmalıdır.



Şekil 6.1 Eclipse geliştirme aracı için hazırlanan eklenti ile mimarinin yapılandırılması

Geliştirme ve test için kullanılan değişkenler aşağıdaki gibi seçilmiştir.

- Giriş Sayfası: LOGIN.guiml
- Guiml Test URLi: http://localhost:8080/GMAjax/index.html
- Dil: tr
- Modellemede Kullanılabilecek Diller: tr, en

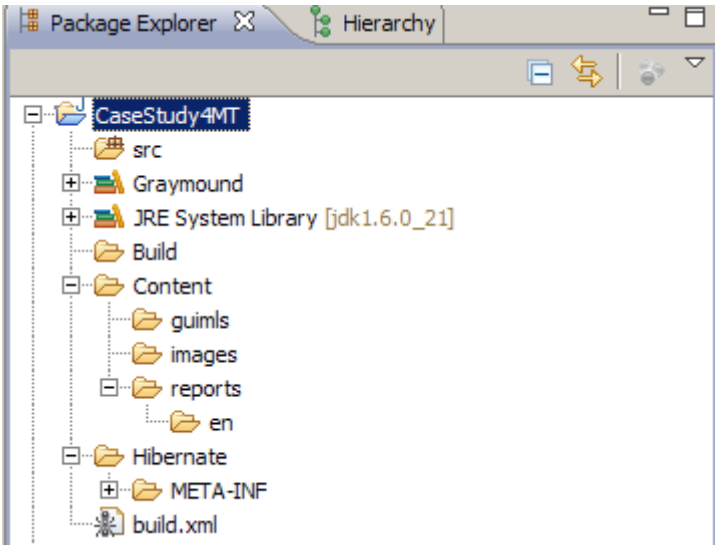
Sunucu tarafındaki değişkenler aşağıdaki gibi seçilmiştir.

- Uygulama İçerik Sağlayıcıları; gerçeklenecek sistemin sadece masaüstü ve web uygulaması olarak çalışması istendiğinden *JAVA* ve *JSON* seçilmiştir.
- Önbellek yapılandırması; varsayılan değerler kullanılmıştır.

- Sunucu bağlantı adresi ve kullanıcı ayarları; varsayılan değer kullanılmıştır.
- Veri Kaynağı; Derby VTYS kullanıldığı için “Derby” ayarları seçilmiştir. Buna bağlı olarak ürün geliştirmede *Hibernate* yapılandırması için Derby ayarları seçilecektir.
- Servis Çalıştırıcı; varsayılan (*GMServiceExecuterDefault*) seçilmiştir.
- Sunucu başlatma; *GMServerStartupHibernate* seçilmiştir.

6.1.2 Yeni Proje Oluşturulması

Yeni bir ürün geliştirmek için Graymound geliştirme aracı üzerinde yeni bir Graymound projesi oluşturmak gerekmektedir. Örnek çalışma için CaseStudy4MT adını verilen yeni bir Graymound Projesi oluşturulmuştur. Bu işlem sonucunda ürün hattı mimarisinden türeterek bir alt ürün mimarisi oluşturulmuştur. Şekil 6.2’de projenin elemanları ve klasör yapısı görülmektedir.



Şekil 6.2 Graymound örnek proje yapısı

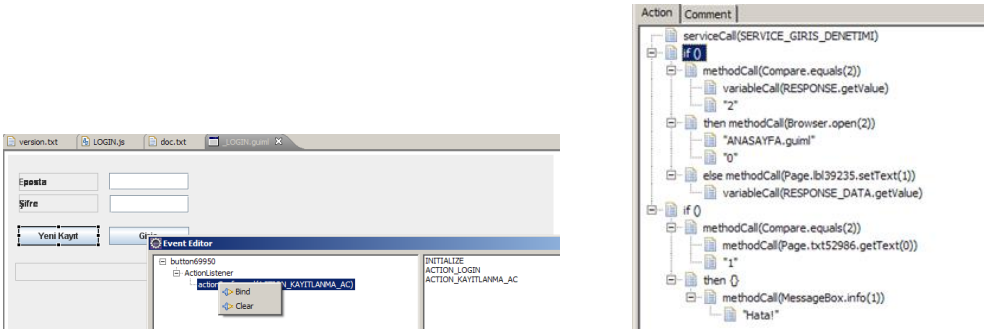
6.2 Gereksinim Analizi Varlıklarının Kullanımı

Uygulama mühendisliğinde sistem analizi için alan mühendisliği kapsamında yapılan gereksinim analizi çalışmaları kullanılır. Ana yazılım varlıklarının analizi, bu varlıkları kullanan ürünler için de geçerlidir. Burada bileşen, servis seçme ve ürüne özgü kod geliştirme için analiz dokümanı hazırlanır.

Sistemin tasarım ve gerçeklemesinde ilk önce sistem analistleri sayfaların modellemesini yaparlar. Şekil 6.3'de kullanıcı giriş sayfası için Graymound editörü ile oluşturulan model dosyası yer almakta ve sayfada olması istenen ön yüz bileşenleri ve bunların birbirleriyle bağlantısı gösterilmektedir.

Graymound, oluşturulan model dosyasından otomatik olarak analiz dokümanı oluşturmaktadır. Ürünün gereksinim analizi süreci Şekil 5.6'da gösterildiği şekilde sistem analistlerince tamamlanır.

Örnek ADK uygulamasındaki kullanıcı sayfaları kayıt, giriş ve işlem olmak üzere üç tanedir. Uygulamanın gereksinim analizi sürecinde oluşturulan modeli şekil 5.6'daki gibidir.



Şekil 6.3 Örnek uygulamada bileşenlerin modellenmesi

Bu sayfaların analiz dokümanları, sayfaların model dosyaları üzerine sağ tıklayınca açılan menüden "Graymound Requirement Analysis" sekmesi seçilerek oluşturulabilmektedir.

Gereksinim analizi sürecinin çıktısı olan model dosyası, geliştirme aşamasında girdi olarak ele alınır ve buna göre gerçekleme yapılır.

6.3 Uygulama Mühendisliğinde Geliştirme Süreci

Bu bölümde uygulama mühendisliği aşamasında geliştiricilerin geliştirme yaparken ana varlıkları nasıl kullandığı gösterilmiştir.

6.3.1 Ön Yüz Bileşenlerinin ve Değişkenlerin Seçilmesi

Alan mühendisliği kapsamında geliştirilen istemci tarafındaki ana varlıkların kullanımı, ön yüz bileşenlerinin seçilmesi ile oluşturulan GUIML adı verilen model dosyaları ile

sağlanır. Ürün geliştiriciler, geliştirme aracının sağladığı palette yer alan ön yüz bileşenlerini seçerek ve yapılandırarak sunum katmanını oluştururlar.

Oluşturulan guiml uzantılı bu model dosyası daha sonra *içerik sağlayıcı*dan gelen isteğe göre işlenerek kod üreteci ile JSON, JavaScript ve JAVA objelerine çevrilecektir.

Örnek çalışmada geliştirilecek uygulamanın sunum katmanı için **gereksinim analizi** sürecinde, hazırlanan model dosyasında gerekli değişiklikler yapılarak modelin tamamlanması ve gerekli değişkenliklerin bağlanması, yani değişkenlerin seçilmesi işlemleri yapılmıştır. Bunun için Guiml Editör'ün sağladığı alana özgü dil kullanılarak ön yüzler gerçeklenmiştir.

Bu model dosyasında kullanıcı giriş servisi çağrıldıktan sonra gelen sonuç değerine göre yönlendirme yapılmış veya hata mesajı verilmiştir. Örnek uygulamada diğer istemci tarafındaki değişkenlikler için varsayılan değerler seçilmiştir. Ürüne özgü bileşen geliştirme veya mevcut bileşenleri genişletme ihtiyacı olmamıştır.

6.3.2 İş Katmanındaki Değişkenliklerin ve Servislerin Seçilmesi

Oluşturacağımız yeni uygulama için istemci tarafındaki özellikleri seçtikten sonra orta katman bileşenleri ve servisleri seçilir. Alan mühendisliği sürecinde hazırlanmış olan iş katmanı ana varlıkları için değişim noktalarına karşılık gelen değişkenler seçilir.

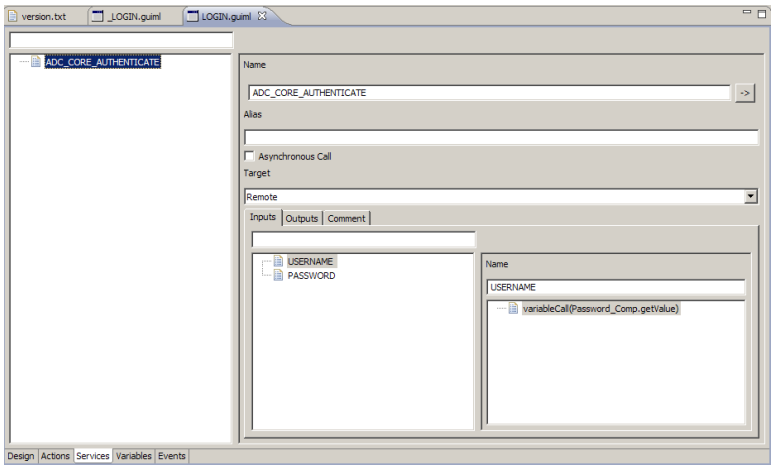
Kullandığımız Yazılım Ürün Hattında en önemli geliştirme adımlarından biri ürüne özgü servislerin geliştirilmesidir. Ürün Hattı her ne kadar kapsam içerisinde ortak kullanılan servisler sağlasa da ürünler çoğu zaman kendilerine özgü servislere ihtiyaç duymaktadırlar.

Kullanıcı kayıt ve kullanıcı giriş sayfaları için Çizelge 6.1'deki ortak servisler belirtilen parametreler ile kullanılmıştır. Burada kullanılan ortak servisleri uygulamaya göre değiştirmek için kendi servislerimizi yazabiliriz. Ürün hattı mimarisi bize bu servisleri genişletme seçeneği sunmaktadır.

Çizelge 6.1 Seçilen servislerin değişim noktalarına karşılık gelen parametreler

SAYFA	SERVİS ADI	PARAMETRELER, DEĞİŞKENLER
Kullanıcı Kayıt Sayfası	ADC_CORE_CREATE_USER	Variable.call(“name”).getValue() Variable.call(“username”).getValue() Variable.call(“email”).getValue() Variable.call(“password”).getValue()
Kullanıcı Giriş Sayfası	ADC_CORE_AUTHENTICATE	Variable.call(“name”).getValue() Variable.call(“username”).getValue() Variable.call(“password”).getValue() Variable.call(“language”).getValue() Variable.call(“channel”).getValue() Variable.call(“call_id”).getValue()
Yeni İşlem Sayfası	ADC_CORE_PROCESS_EXECUTE	Variable.call(“process_code”).getValue() Variable.call(“process_key”).getValue() Variable.call(“pass_execution”).getValue() Variable.call(“amount”).getValue() Variable.call(“amount_code”).getValue()

Şekil 6.4’de Eclipse tabanlı editör ile model üzerinde servislerin seçilmesi ve parametrelerinin girilmesi gösterilmiştir.



Şekil 6.4 Farklı uygulama tiplerinden çağrılan servislerin değişim noktalarının bağlanması ve parametrelerinin seçilmesi

Şekilde görülen “Variables” sekmesinde, servis sekmesinde girilen parametrelerin değerleri yönetilmekte ve parametreler bu değişkenlerden alınmaktadır.

Sunucu tarafındaki servis çalıştırıcısı, servis sınıfları, mesaj verisi, önbellek gibi değişim noktaları için kurulum ile birlikte gelen varsayılan değerler kullanılmıştır.

Sunucu tarafındaki bir diğer değişim noktası olan *istemci sağlayıcı* bu bölümün son başlığı olan uygulamanın çalıştırılması başlığı altında anlatılmıştır.

6.3.3 Ürüne Özgü Servis Geliştirme

Mevcut servisler gerçeklenecek yazılım sisteminin ihtiyaçlarını karşılamıyorsa yeni servisler yazılarak mimari genişletilebilir. Servis geliştirme işlemi için ilk önce servisin kayıt edilmesi gerekmektedir. Bunun için *GMServiceConfiguration. xml* dosyasına, eklenecek olan servisin bulunduğu sınıfın adı girilir. Daha sonra servisin yapacağı işin kodlanmasına geçilir.

Örnek ADK uygulaması için kullanılan kullanıcı giriş servisini değiştirmek istediğimizi düşünelim. Oluşturduğumuz yeni servis Java Annotation[1] teknolojisini kullanarak tanımlanmaktadır. Servis yükleyiciler sunucu açılırken, işaretlenmiş kodları kullanarak ürüne özgü servisleri kaydeder.

EKB-1’de giriş denetimi yapan servis ve servis parametrelerine ait kaynak kod görülmektedir.

6.4 Ürün Testi

Geliştirilen yeni ürünün testi, ürün hattının sağladığı yeniden kullanılabilir yazılım test varlıkları ile daha hızlı ve kolay bir şekilde yapılabilmektedir. Sağlanan test aracı ile uygulama test ayarları ile çalıştırılarak ön yüz modelleme, seçilen servislerin birim testi ve uyum testi bir test aracı ile yapılabilmektedir.

Ürün testi için seçilen içerik sağlayıcıya göre “test” parametresinin ayarlanması gerekmektedir. Böylelikle uygulama test kipinde açılacaktır. Test kipinde açılması

[1] Java Annotations: Oracle Sun Java Tutorials, Annotations, download.oracle.com/javase/tutorial/java/javaOO/annotations.html

demek kullanıcı girişi, yetkilendirme işlem yapabilme gibi işlemlerin taklit edilerek (mocking) çalıştırılması anlamına gelmektedir. Web istemcisi için test kipinde açılan bir sayfanın erişim linki aşağıda verilmiştir.

http://localhost:8080/GMAjax/index.html?test=true&firstPage=LOGIN.js&loginPage=LOGIN.guiml&language=tr

Seçilen model istenirse bir web tarayıcısıyla web uygulaması olarak veya Java Swing Kütüphanesi kullanılarak masaüstü uygulaması olarak test edilebilir. Test durumları için hazırlanması gereken bağlantı ve kullanıcı yetkilendirme işlemi, YÜH'nin test ayarları seçilerek tekrar kullanılmış olur.

6.5 Uygulamanın Tipinin Seçilmesi ve Çalıştırılması

Yeni uygulama modellenip testi yapıldıktan sonra yeni ürünler oluşturmak için seçilmesi gereken en önemli özellik *içerik sağlayıcı*dır. Alan mühendisliği kapsamında sunulan bu özellik için sağlanan değişkenler daha önce de bahsettiğimiz Java, JavaScript – JSON, SOAP ve Report içerikleridir. Bu değişkenin seçilmesi yazılım sisteminin taşınmasından sonra yapılabildiği için Graymound bir **DSPL** olarak kabul edilebilir. Yani, YÜH'den işletim platformuna göre ürün oluşturulabilmektedir.

İçerik sağlayıcılardan biri seçilerek yeni ürün tamamlanmış olunur. Seçilen içerik sağlayıcı değişkenine göre Graymound Yazılım Ürün Hattı, yine alan mühendisliği kapsamında geliştirilen Alana Özgü Dil ve bu dile ait kod üreteçleri ile model dosyasından otomatik kod üretimini yapar.

Yeni ürünün kodları çalışma anından otomatik olarak oluşturulur. Çizelge 5.1 'de İçerik Sağlayıcı özelliği için değişkenler ve değişkenlik bağlama zamanı bilgileri verilmektedir.

İçerik sağlayıcı, gelen isteğin içerik bilgisine bakarak seçilir. Her içerik sağlayıcı için sunucu tarafındaki varlıklardan biri olan içerik sağlayıcı Java Servlet sınıfları bulunmaktadır.

Örnek uygulama için, Çizelge 6.2'deki URLleri kullanılarak içerik sağlayıcıya göre yeni bir ürün oluşturulabilir.

Çizelge 6.2 Örnek sistem için seçilen içerik sağlayıcıları

İÇERİK SAĞLAYICI DEĞİŞKENİ	DEĞİŞKENİN SEÇİMİ
Java	Host:port/GMJava/
JSON – JavaScript	Host:port/GMAjax/index.html

Örnek yazılım uygulaması için *içerik sağlayıcı* değişkeni seçilmesi ile birlikte çalışma anında gerekli olan sunum ve iş katmanı değişkenleri de bağlanarak Java Swing ve Ajax web uygulaması elde edilmiştir. EKB-2'de farklı dil ayarlarına sahip bu ürünlere ait ekran görüntüleri yer almaktadır.

Görüldüğü üzere platform çoklayıcı YÜH modeli ile gerçeklenen örnek ADK yazılım sistemi, hem masaüstü hem de web tarayıcısından çalıştırılabilmektedir.

6.6 Üretim Planı ve Kısıtları

Geliştirilen YÜH'nin ürün gerçekleme sırasında nasıl kullanılacağının belirlenmesi ve belgelenmesi gerekmektedir. Organizasyon, bunun için Graymound ile yazılım geliştirme adımlarını içeren bir rehber sunmaktadır.

Üretim planına göre; müşteri ile yazılım sistemi gerçekleme üzerinde anlaşıldıktan sonra, sistem analistleri müşterinin isteklerine göre gereksinim analizini yaparak model dosyasını oluşturur. Geliştiriciler model dosyasında gerekli bileşenleri ve servisleri bağlayarak ürünü gerçeklerler. Sunulan test aracı ile ürün doğrulanır ve taşıma işlemi için yazılım sistemi hazır hale getirilir.

Alan mühendisliği çalışmalarının bir çıktısı olarak elde edilen üretim planına göre bir **yazılım fabrikası** hazırlanmış olunur. Böylelikle üretkenlik ve geliştirme hızı artmaktadır.

ADK uygulamalarında YÜH kullanımının getirdiği faydaların yanında, belirli bir mimariye bağımlı olmak ve her farklı gereksinim için ürüne özgü bileşenler geliştirmenin zorluğu birer üretim kısıtlaması olarak düşünülebilir.

6.7 Yeniden Kullanım Ölçümleri

Bir yazılım ürün hattı mimarisi için yeniden kullanımın ne oranda olduğunu bilmek önemlidir. Yazılım varlıklarının ne kadarının yeniden kullanıldığı, ürüne özgü kodlara ne kadar ihtiyaç duyulduğu, ne kadar dokümantasyon yapıldığı önemli ölçütlerdir [4], [38]. Buna bağlı olarak çalışmamızda kullanılacak olan metrikler aşağıda listelenmiştir.

- Üretilen html ve JavaScript dosya sayısı
- Üretilen sınıf sayısı
- Üretilen kod satır sayısı
- Gerçeklenen müşteri gereksinimi sayısı
- Üretilen doküman sayısı
- Değişkenlerin Seçilmesi Harcanan İş Gücü

Graymound YÜH, sunum katmanının tamamını, istemciye göre otomatik kod üreten içerik sağlayıcı özelliği ile üretebilmektedir. İş katmanı için sağladığı ortak servisler, ürün ihtiyaçlarına uymuyorsa ürüne özgü servis geliştirilebilmektedir. Veri katmanı için ise yeniden kullanım sağlanmaktadır. Ancak her ürün kendi veri tabanı sistemini kullanmak istediğinde veri katmanı için yeniden kullanım azalmaktadır.

Örnek yazılım gerçeklemesinde yer alan ürünlerden web içeriği ile çalışan ürün için Çizelge 6.3'deki değerlere ulaşılmıştır. Otomatik olarak üretilen dosyalar ve içerikleri EKB-3'de yer almaktadır.

Çizelge 6.3 Web ADK yazılım ürünü için otomatik üretilen ve gerçeklenen yazılım varlığı değerleri

SAYFA	ÜRETİLEN DOSYA SAYISI (guiml, HTML, Javascript)	TOPLAM ÜRETİLEN LOC	ÜRÜNE ÖZGÜ ~LOC	MÜŞTERİ GEREKSİNİM / ÜRETİLEN DOKÜMAN SAYISI	YENİDEN KULLANILAN SERVİS SAYISI
Kullanıcı Kayıt Sayfası	1 Javascript 1 GUIML	744	-	2 / 1	2

Kullanıcı Giriş Sayfası	1 Html 1 Javascript 1 GUIML 2 Hibernate nesnesi	698	375	2 / 1	1
Yeni İşlem Sayfası	1 Javascript 1 GUIML	880	-	2 / 1	3

Örnek uygulamada yer alan ürünlerden Java içeriği ile çalışan masaüstü ürünü için Çizelge 6.4'deki değerlere ulaşılmıştır.

Çizelge 6.4 Masaüstü ADK yazılım ürünü için otomatik üretilen ve gerçeklenen yazılım varlığı değerleri

SAYFA	ÜRETİLEN GUIML, BAĞLANAN SINIF SAYISI	ÜRETİLEN ~LOC	ÜRÜNE ÖZGÜ ~LOC	MÜŞTERİ GEREKSİNİM / ÜRETİLEN DOKÜMAN SAYISI	YENİDEN KULLANILAN SERVİS SAYISI
Kullanıcı Kayıt Sayfası	1 model dosyası 13 bileşen bağlanması	1100	-	2 / 1	2
Kullanıcı Giriş Sayfası	1 model dosyası 17 bileşen bağlanması 2 Hibernate nesnesi	950	375	2 / 1	1
Yeni İşlem Sayfası	1 model dosyası 18 bileşen bağlanması	1200	-	2 / 1	3

Gerçeklenen Müşteri Gereksinim Sayısı; uygulamanın büyüklüğüne göre değişebileceği gibi burada gösterilen örnek uygulamada 6 olarak alınmıştır. Bunlardan 3 tanesi fonksiyonel gereksinimlerdir. 3 tanesi yeniden kullanılan performans gereksinimleridir.

Bu metrikler dışında yeniden kullanılacak servislerin ve model dosyalarının örnek sistem gerçeklemesinde ne oranda kullanıldığı da metrikler arasına alınmıştır. Servislerin yeniden kullanım oranı ölçütü, Servis Tabanlı YÜHler için bir metrik olarak kullanılabilir.

Kullandığımız YÜHM yaklaşımında geliştirme sürecine etki eden en önemli etken olan değişkenlerin seçimi için ise aşağıdaki değerlere ulaşılmıştır.

Değişkenlerin Seçilmesi için Harcanan İş Gücü: 3 adet model dosyası geliştirilmiştir. Bu model dosyaları birincisi olan kayıt sayfası için 5 adet ön yüz bileşeni seçilmiş, servis kısmında 1 adet servis seçilmiştir. İkincisi olan giriş sayfası için 3 adet bileşen seçilmiş ve servis kısmında 1 adet servis seçilmiştir. İşlem sayfası için 5 adet ön yüz bileşeni ve 1 adet servis seçilmiştir.

Kullanılan yeni işlem ve kayıt servisleri ortak servisler olduğu için orta katmanda bir geliştirmeye ihtiyaç duyulmamıştır. Ürüne özgü kullanıcı giriş servisi yeniden yazılmıştır. Buradan yola çıkarak sırasıyla sunum katmanı ve iş katmanı için yeniden kullanım oranları;

Üretilen LoC / Toplam Loc ~= 4750 / 6500 = %73 ve

Yeniden Kullanılan Servis Sayısı / Toplam Servis Sayısı, 6 / 8 = %75 olarak alınmıştır.

Ancak birçok gerçek uygulamada daha fazla ürüne özgü düzenlemeler gerekmekte ve ek olarak kodlar yazılmaktadır. Graymound ile bir yazılım sistemin gerçeklenirken ürüne özgü geliştirme maliyeti hesaplanırken bir takım parametrelere dikkate alınır. Bunlar; veri tabanı erişim katmanı çağrılarının sayısı, doğrulama sayısı, servis ve web servis çağrılma sayısı, ön yüz bileşen sayısı ve entegre edilmesi gereken eski kod sayısıdır.

BÖLÜM 7

SONUÇ

Bu tez çalışmasında yazılım üretkenliğini artırmak amacıyla geliştirilen Servis Tabanlı bir Yazılım Ürün Hattı modeli ile farklı istemci platformlarında çalışabilen uygulamalar geliştirmek için bir model sunulmuştur. Bu model daha önceden geliştirilen Graymound YÜH mimarisini referans alarak genel bir şekilde açıklanmıştır. Daha sonra ADK yazılımları alanı için bu modelin bir gerçeklemesi olan örnek YÜH'de, alan mühendisliği ve ürün geliştirme aşamaları gösterilmiştir. Graymound YÜH'nin istemci tipi içerik sağlayıcı değişkeni seçilerek farklı istemciler için yazılım ürünleri elde edilmiştir.

Genel anlamda yazılım geliştirmede YÜHM yaklaşımı kullanımının yazılım üretkenliğini artırdığı birçok çalışmada gösterilmiştir. Yazılım Ürün hattı kullanılarak geliştirilen ürünler için yeniden kullanım metrikleri uygulanarak ne kadar hızlı, kolay, kaliteli kod, analiz ve test durumları geliştirildiği gözlemlenebilmektedir. Bölüm 6.7'de gösterildiği gibi yeni bir yazılım sistemi gerçeklemesinde kodun ve dokümanın büyük çoğunluğu YÜH tarafından oluşturulmaktadır. Böylelikle yeniden kullanım ve yazılım üretkenliği artmaktadır.

Sadece YÜH mimarisinin kullanıldığı düşünüldüğünde, **içerik sağlayıcı**ya göre farklılaştırılan her bir ürün için değişkenliklerin seçilmesi ve ürüne özgü geliştirme yapılması gerekecektir. Bu da ürün gerçekleme süresini arttıracaktır.

Sunulan model 3 katmanlı mimariler için Model Güdümlü geliştirme ve Servis Tabanlı mimari kullanımı ile bu özelliğin farklılaştığı yerlerde mimari dinamik olarak yapılandırılabilecek şekilde geliştirilmiştir. Böylelikle her bir ürün için baştan Uygulama Mühendisliği adımlarının yapılmasına gerek duyulmamaktadır. Sonuç olarak, platform çoklayıcı modele göre geliştirilen **Servis Tabanlı YÜH** mimarisinin sağladığı İçerik

Sağlayıcı özelliğine ait değişkenliğin çalışma anında dinamik olarak seçilebilmesi ürün gerçekleme süresini diğer YÜH yaklaşımlarına göre kısaltmaktadır.

7.1 Kalite ve Riskler

Graymound ile YÜHM yaklaşımının kullanılması ile yazılım geliştirmede aşağıda listelenen kalite açısından büyük fayda sağlanmaktadır.

- Test edilmiş ADK bileşen ve servislerinin tekrar kullanılması
- Başka yazılım ürünlerinde kullanılan yazılım varlıklarının kullanımı
- Kaynak kodun büyük çoğunluğunun otomatik olarak üretilmesi
- Üretilen kodun standartlara uygun olması
- Gereksinim analizi sürecinde geliştirilen model dosyasının geliştirme sürecinde de kullanılması ile hata oranın azaltılması

Yazılım Geliştirme Riskleri: ADK yazılımları geliştirmek için platform çoklayıcı bir YÜH yaklaşımı kullanımı, üretim hızının ve kalitenin artması gibi faydaların yanında bir takım riskleri de beraberinde getirmektedir. Geliştirilecek ADK yazılımında mevcut bileşenler gereksinimleri karşılamıyorsa ürüne özgü kod geliştirme geleneksel yönteme göre daha maliyetli olabilir.

7.2 Araştırma Konuları

Graymound için ileride yapılması planlanan ürün hattı mimarisinde bir değişiklik olduğunda bunun tüm ürünlerde diğer özelliklerin düzgün çalışmasının test edilmesi için ana varlıklara *regresyon test* araçları eklenmesi planlanmaktadır. Veri tabanı işlemleri testi için DBUnit, uyum testi için EasyMock kütüphanelerinin Graymound ana varlıklarına eklenmesi düşünülmektedir.

Test araçlarının sağlanması dışında YÜH mimarisini sadece SOA yaklaşımına bağımlı kılmamak için İlgiye Dayalı Yazılım Geliştirme yöntemine göre de dinamik yapılandırma sağlamak amaçlanmıştır. Böylelikle “istemci tipi” değişkenliğinde olduğu gibi diğer değişkenlik noktaları için de dinamik değişken bağlama sağlanması düşünülmektedir. Bu konuyla bağlantılı DSPL yaklaşımı için, çalışma zamanı değişkenlik yönetimi, dinamik

kütüphane yükleme, yansıma, dinamik değişkenlik yönetiminin modellenmesi ve İlgiye Dayalı Programlama araştırma konuları üzerinde çalışmalar yapılması faydalı olacaktır. Dinamik Yazılım Ürün Hatları, son 5 yılda yapılan çalışmalar ile iş süreçlerinin yönetildiği kurumsal uygulamalar için en uygun yeniden kullanım yaklaşımlarından biri olarak dikkat çekmektedir.

KAYNAKLAR

[1] Ezran, M., Morisio, M. ve Tully, C., (2002). "Practical Software Reuse", First Edition, Springer, London.

[2] Frakes, W.B. ve Kang, K., (2005). "Software Reuse Research: Status and Future", IEEE Transactions on Software Engineering, 31(07):529-536.

[3] Crespo, S., Fontuora, M.F. ve Lucena, C.J., "Using Viewpoints, Frameworks, and Domain Specific Languages to Enhance Software Reuse", http://fontoura.org/papers/erw98.pdf, 30 Ocak 2010.

[4] Almeida, E.S., Alvaro, A., Garcia V. C., Mascena, J. C. C. P., Buregio, V. A. A., Nascimento, L., Lucredio, D. ve Meira, S. L., (2007). "Component Reuse in Software Engineering", First Edition, C.E.S.A.R e-books.

[5] Shaw, M. ve Garlan, D., (1996). "Software Architecture: perspective on an emerging discipline", Second Edition, Prentice Hall Publishing, New Jersey.

[6] Gamma, E., Helm, R., Johnson, R. ve Vlissides, J., (1994). "Design Patterns: Elements of Reusable Object-Oriented Software", First Edition, Addison-Wesley Professional, Boston.

[7] Sommerville, I., (2010). "Software Engineering", 9th Edition, Pearson, Boston.

[8] Torchiano, M. ve Morisio M., (2004). "Overlooked Aspects of COTS-Based Development", IEEE Software, 21(2):88-93.

[9] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M, Palm, J. ve Griswold, W.G., (2001). "Getting Started with AspectJ", Communications of the ACM, 44(10):59-65.

[10] Heineman, T. ve Councill, B., (2001). "Definition of a Software Component and its Elements", syf 5-20; Derleyen Heineman, T. ve Councill, B., (2001). "Component-Based Software Engineering: Putting the Pieces Together", Addison-Wesley, Boston.

[11] Taylor, N., Nenad, M. ve Dashofy, E. M., (2009). "Software Architecture: Foundations, Theory, and Practice", First Edition, John Wiley & Sons, New Jersey.

[12] Mernik, M., Heering, J. ve Sloane, A. M., (2005). "When and how to develop domain-specific languages", ACM Computing Surveys, 37(4):316-344.

[13] Kelly, S. ve Tolvanen, J.P., (2008). "Domain-Specific Modeling: Enabling Full Code Generation", First Edition, John Wiley & Sons, New Jersey.

[14] Clements, P. ve Northrop, L., (2002). "Software Product Lines: Practices and Patterns", Third Edition, Addison Wesley, Boston.

[15] Kang, K. C., Lee, J. ve Donohoe, P., (2002). "Feature-Oriented Product Line Engineering", IEEE Software, 19(4):58-65.

[16] Wikipedia, Feature-Oriented Programming, http://en.wikipedia.org/wiki/Feature-oriented_programming, 20 Mart 2011.

[17] Atkinson, C., Bayer, J. ve Muthig, D., (2000). "Component-Based Product Line Development: KobrA Approach", First Software Product Line Conference, 28-31 Ağustos 2000, Denver, 289-309.

[18] Carnegie Mellon University Software Engineering Institute, A Framework for Software Product Line Practice Version 5.0, www.sei.cmu.edu/productlines/frame_report/index.html, 15 Mayıs 2011.

[19] Linden, F., Schmid, K. ve Rommes, E., (2007). "Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering", Springer, New York.

[20] Garg, A., Critchlow, M., Chen, P., Westhuizen, C. ve Hoek, A., (2003). "An Environment for Managing Evolving Product Line Architectures", International Conference on Software Maintenance ICSM'03, 22-26 Eylül 2003, Amsterdam, 358-367.

[21] Çatal, Ç, (2010), "Geleneksel Yazılım Mühendisliğinden Alana Özel Yazılım Mühendisliğine Doğru", Akademik Bilişim, 10-12 Şubat 2010, Muğla.

[22] Sinnema, M. ve Deelstra, S., (2006), "Classifying variability Modeling techniques", Elsevier Information and Software Technology, 49:717-739.

[23] Kahraman, E., İpek, T., İyidir, B., Bazlamaçcı, C.F. ve Bilgen, S., (2009), "Bileşen Tabanlı Yazılım Ürün Hattı Geliştirmeye Yönelik Alan Mühendisliği Çalışmaları", 4. Ulusal Yazılım Mühendisliği Sempozyumu, 8-10 Ekim 2009, İstanbul,283-287.

[24] Charles, W.K., (2006), Introduction to the Emerging Practice of Software Product Line Development, http://www.methodsandtools.com/archive/archive.php?id=45p4, 23 Ocak 2011.

[25] Linden, F. v. d., (2009). "Applying Open Source Software Principles in Product Lines", UPGRADE CEPIS The European Journal for the Informatics Professional, Libre Software for Enterprises, 10(3):32-40.

[26] Gonzalez, S. T., (2007). Feature Oriented Software Product Lines, Doktora Tezi, University of the Basque Country Computer Science, San Sebastian.

[27] Ye, H. ve Liu, H., (2005). "Approach to modelling feature variability and dependencies in software product lines", IEE Proceedings – Software, 152(3):101-109.

[28] Anastasopoules, M. ve Gacek, C., (2001). “Implementing Product Line Variabilities”, Symposium on Software Reusability, 18-20 Mayıs 2001, Toronto, 109-117.

[29] Ishida, Y. (2007), “Software Product Lines in Enterprise System Development”, 11th International Software Product Line Conference, 10-14 Eylül 2007, Kyoto, Japan, 44-53.

[30] Altıntaş, N. İ., Surav, M., Keskin, O. ve Çetin, S., (2005). “Aurora Software Product Line”, 2. Ulusal Yazılım Mühendisliği Sempozyumu, 22-24 Eylül 2005, Ankara.

[31] Balzerani, L., Ruscio, D., Pierantonia, A. ve Angelis, G., (2005). “A Product Line Architecture for Web Applications”, ACM Symposium on Applied Computing, 13-17 March 2005, Santa Fe, New Mexico, 1689-1693.

[32] Software Product Line Conferences, Hall of Fame, http://splc.net/fame.html, 21 Mart 2011.

[33] Parlakol, N. B., (2010). A Test Oriented Service and Object Model for Software Product Lines, Yüksek Lisans Tezi, ODTÜ Fen Bilimleri Enstitüsü, Ankara.

[34] Gimenes, I., Fantinato, M. ve Toledo, M.B., (2008). “A Product Line for Business Process Management”, 12th Software Product Line Conference, 8-12 Eylül 2008, Limerick, 265-274.

[35] Krut, R. ve Cohen, S., (2008). “Service-Oriented Architectures and Software Product Lines – Putting Both Together”, 12th Software Product Line Conference, 8-12 Eylül 2008, Limerick.

[36] Lee, J. ve Kotonya, G., (2010). “Combining Service Orientation with Product Line Engineering”, IEEE Software, 27(3):35-41.

[37] Schmidt, D., Nechypurenko, A. ve Wuchner, E., (2005), “MDD for Software Product-Lines: Fact or Fiction”, 8th International Conference on Model Driven Engineering Languages and Systems, 2-7 Ekim 2005, Montego Bay, Jamaica.

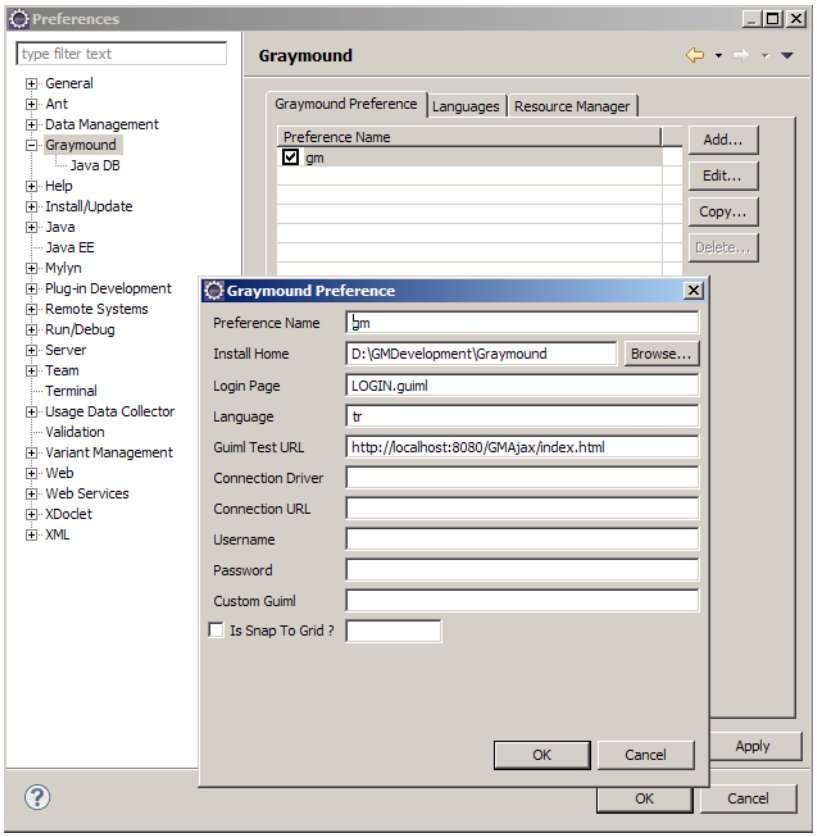
[38] Zubrow, D. ve Chastek, G., (2003). Measures for Software Product Lines, Technical Note CMU/SEI-2003-TN-031.

EK-A

ALAN MÜHENDİSLİĞİ ÇALIŞMALARI

A-1 Geliştirme Ortamı Hazırlanması

Eclipse üzerinde geliştirme ortamının ve ürün hattı mimarisinin kurulumundan sonra Şekil A-1.1'de gösterildiği gibi istenilen değerler girilerek Graymound YÜH mimarisinin ürüne özel bir örneği elde edilmektedir.



Şekil A-1.1 Ürün geliştirme ortamının hazırlanması

Burada, uygulamanın dili, kullanıcı yetkilendirme sayfası, model test etme bağlantı adresi ve veri tabanı bağlantı bilgileri girilebilmektedir.

A-2 YÜH Tarafından Oluşturulan Proje Derleme Dosyası

Alan mühendisliği kapsamında geliştirilen ve kurulum ile birlikte gelen ana yazılım varlıklarından biri olan Graymound örnek projesi için kaynak kod derleme dosyasına ait içerik aşağıdaki gibidir.

CaseStudy4MT/Build.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project default="Build" name="CaseStudy4MT Builder">
<target name="Build">
<jar destfile="${basedir}/Build/CaseStudy4MT.jar">
<fileset dir="${basedir}/bin" includes="**/*.class"/>
</jar>
<jar destfile="${basedir}/Build/CaseStudy4MT.har">
<fileset dir="${basedir}/Hibernate" includes="**/*.hbm.xml"/>
<fileset dir="${basedir}/Hibernate" includes="**/hibernate*.xml"/>
</jar>
<copy file="${basedir}/Build/CaseStudy4MT.har"
toFile="D:\GMDevelopment\Graymound/Server/Hibernate/CaseStudy4MT.har"/
>
</target>
</project>
```

A-3 Sunucu Tarafındaki Değişkenlikler için Varsayılan Değerler

Çizelge A-3.1'de sunucu tarafındaki değişim noktaları için varsayılan değişkenler yer almaktadır.

Çizelge A-3.1 Varsayılan Değişken Değerleri

DEĞİŞİM NOKTASI	VARSAYILAN DEĞİŞKEN
Ürünün Çalışacağı İstemci Tipleri	GMJavaContext GMAjaxContext GMSOAPContext GMReportContext
Servis Çalıştırıcı	GMServiceExecuterDefault
Servis Yükleyici	Db Service Loader
Yetkilendirme	ADC_CORE_AUTHENTICATE servisi
GMCacheLoader	GMCacheLoaderMap
Transaction Coordinator	GMTransactionCoordinatorDefault

EK-B

UYGULAMA MÜHENDİSLİĞİ ÇALIŞMALARI

B-1 Ürüne Özgü Servis Geliştirme

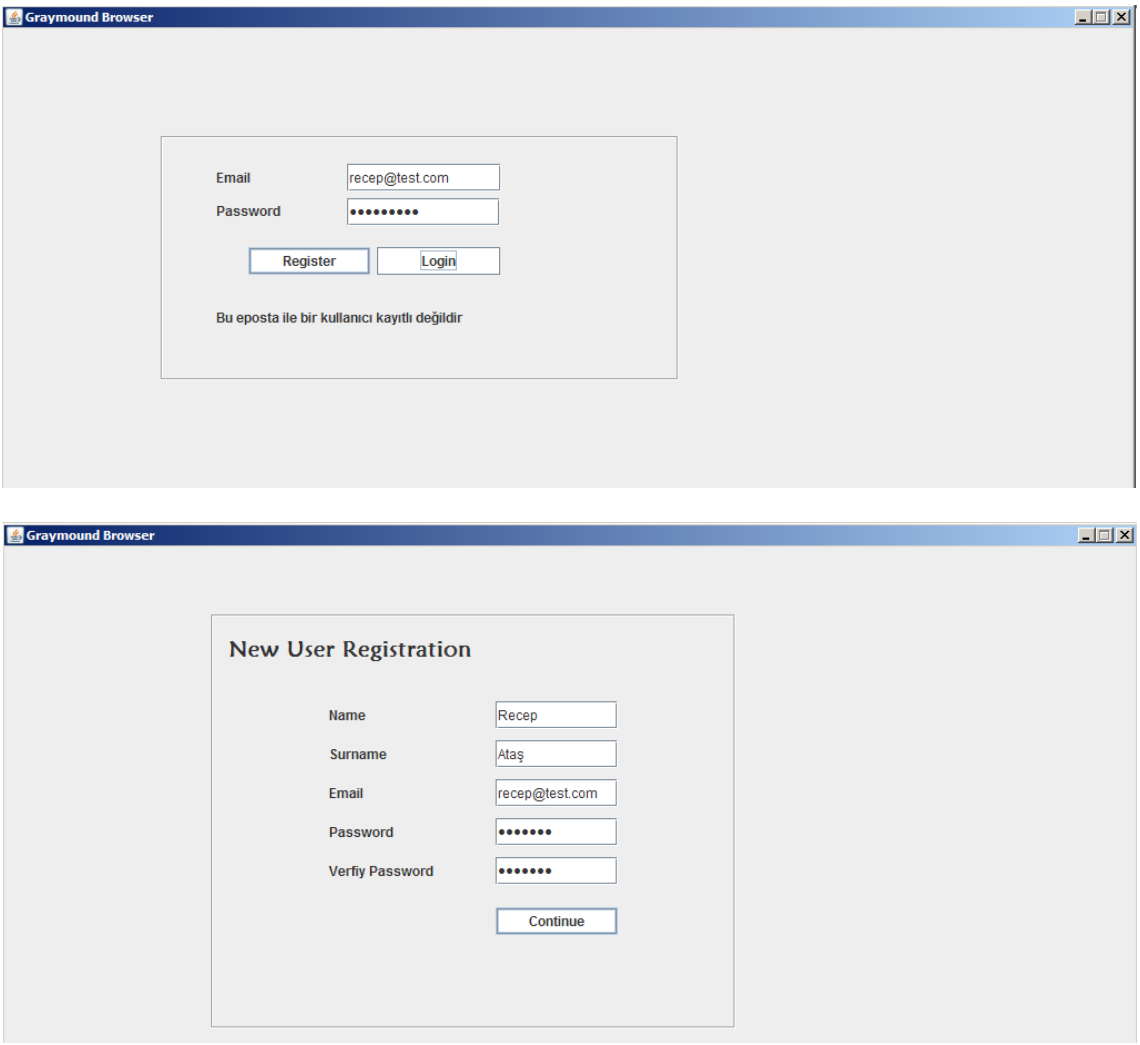
Alan Mühendisliği kapsamında geliştirilen servisler ürün ihtiyaçlarını karşılamıyorsa ürüne özgü servis geliştirme yapılabilir. Bunun için mevcut mimari standartlarına uygun geliştirme yapılmalıdır. Aşağıda kullanıcı giriş denetimi yapan bir servis kaynak kodu yer almaktadır.

```
@GraymoundService(value="SERVICE_GIRIS_DENETIMI",authenticationRequired=false)
@GraymoundServiceParameters(
iParameters = {
@Parameter(name ="EPOSTA", type=ParameterType. SIMPLE),
@Parameter(name ="SIFRE", type=ParameterType. SIMPLE),
@Parameter(name ="GIRIS_SAYISI", type=ParameterType. SIMPLE)
},
oParameters = {
@Parameter(name ="RESPONSE", type=ParameterType. SIMPLE),
@Parameter(name ="RESPONSE_DATA", type=ParameterType. SIMPLE)
}
)
public static GMMap denetle(GMMap iMap){
GMMap oMap = new GMMap();
String eposta = iMap. getString("EPOSTA");
String sifre = iMap. getString("SIFRE");
Session session = DAOSession. getSession("KTSDS");
Criteria criteria=session. createCriteria(Kullanici. class);
Kullanici kullanici= (Kullanici)criteria. add(Restrictions. eq("eposta",
eposta)).uniqueResult();
if (!(kullanici. getSifre().equals(sifre))){
oMap. put("RESPONSE", 0);
oMap. put("RESPONSE_DATA", "Lütfen doğru şifre giriniz.");
return oMap;
}
KTSSession. put("KULLANICI_ID",String. valueOf(kullanici. getId()));
oMap. put("RESPONSE", 2);
return oMap;}
```

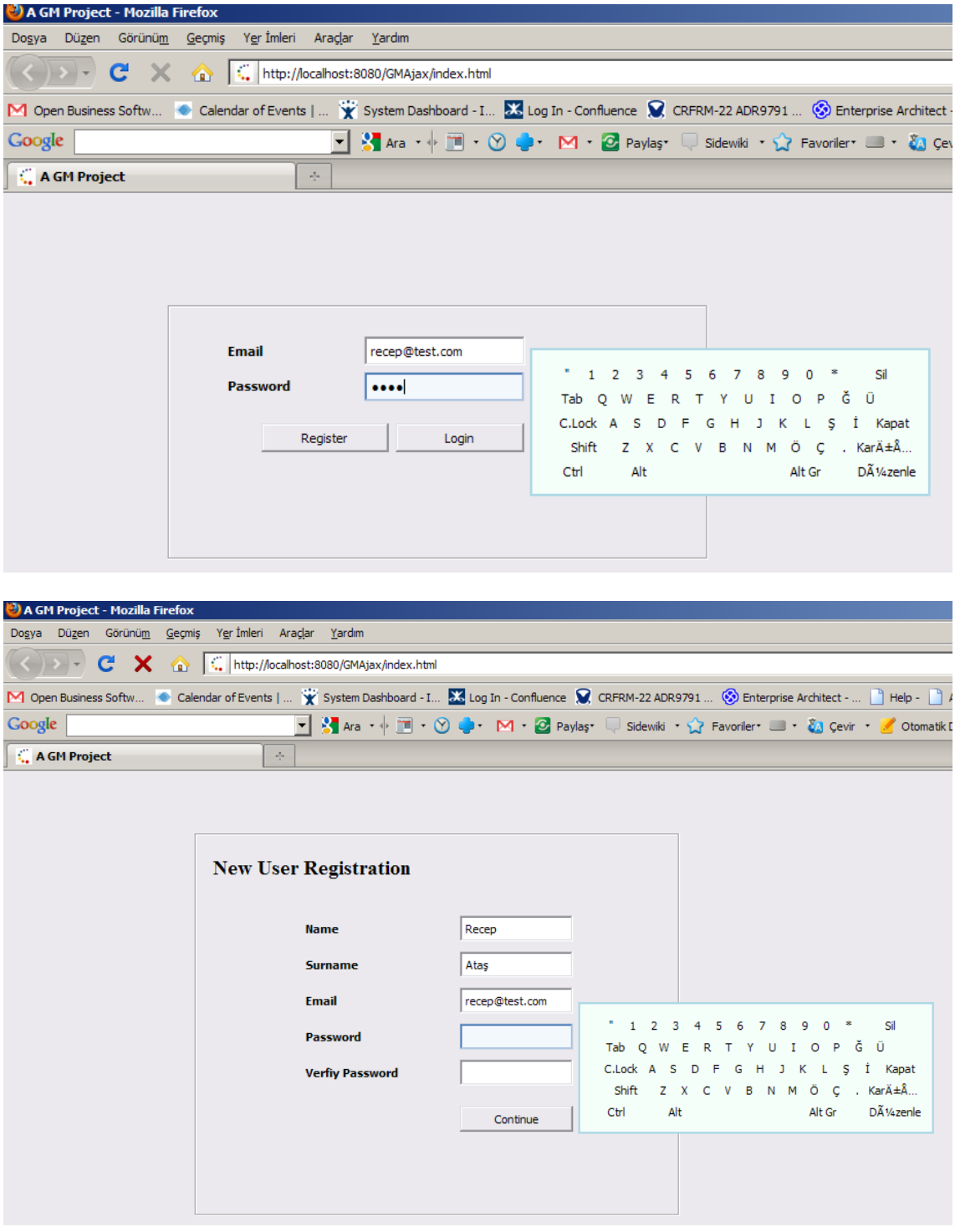
Servis gerçeklenirken tanımlanan Java Annotation nesneleri bu servisin değişkenlerini belirten girdi ve çıktı parametrelerdir. SOA'nın sağladığı yapı ile bu servis, farklı istemci tiplerinden çağrılabilmektir.

B-2 İstemci Platformuna Göre Elde Edilen Ürünler

İstemci platformuna göre farklılaşan yazılım ürünlerinde kullanıcı giriş ve kayıt sayfaları için Java Swing ve Qooxdoo ön yüz sağlayıcıları ile model dosyasından otomatik olarak üretilen sayfaların ekran görüntüleri Şekil B-2.1 ve Şekil B-2.2'de yer almaktadır.



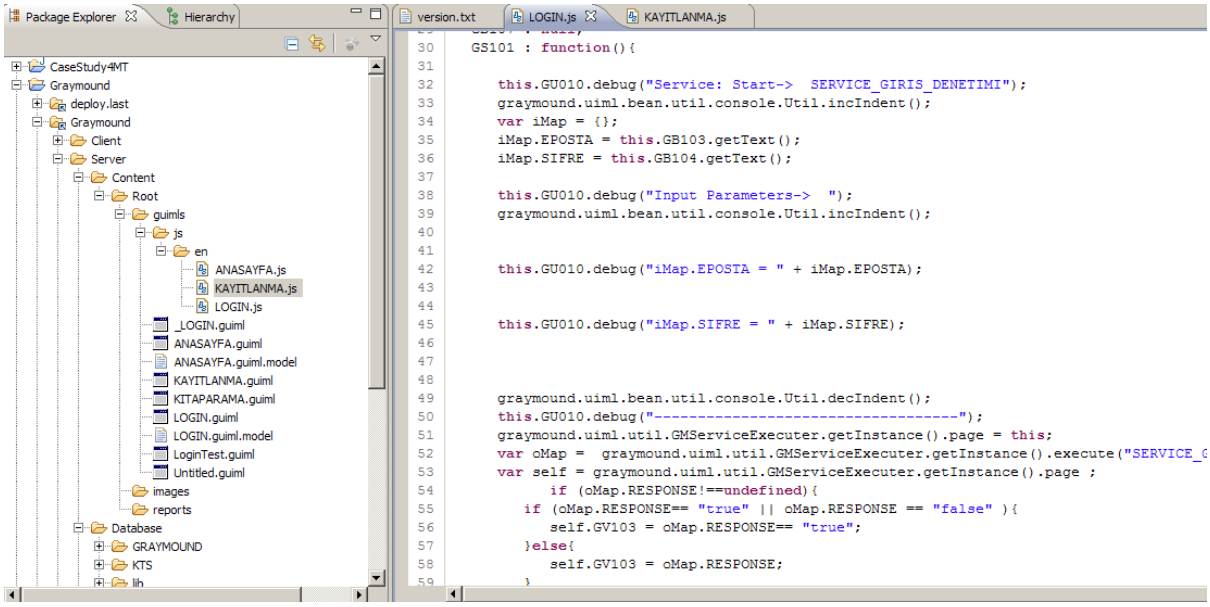
Şekil B-2.1 Masaüstü istemcisi için Java Swing Kütüphanesi ile otomatik oluşturulan ürün



Şekil B-2.2 Web istemcisi için Qooxdoo Kütüphanesi ile otomatik oluşturulan ürün

B-3 Model Dosyasına Girilen DSL Tanımlarına Göre Üretilen Kod

Şekil B-3.1'de model dosyasına girilen DSL tanımlarına ve istemci tipine göre kod üreteçleri tarafından otomatik olarak üretilen dosyalar ve bir Javascript dosyası içeriği gösterilmektedir.



Şekil B-3.1 Web ürünü için otomatik üretilen dosyalar ve içerikleri

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı	: Recep ATAŞ
Doğum Tarihi ve Yeri	: 13.03.1986 Darende
Yabancı Dili	: İngilizce
E-posta	: recepatas@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Lisans	Bilgisayar Mühendisliği	İstanbul Üniversitesi	2008
Lise	Fen	Mersin Yusuf Kalkavan A.L.	2004

İŞ TECRÜBESİ

Yıl	Firma/Kurum	Görevi
2010	OBSS Bilişim	Yazılım Geliştirici - Takım Lideri
2008	OBSS Bilişim	Yazılım Geliştirici

YAYINLARI

Bildiri

1. Recep Ataş, Oya Kalıpsız	Servis Tabanlı Yazılım Ürün Hattı Mimarileri, Elektrik-Elektronik ve Bilgisayar Sempozyumu 2011 Kitabı.