T.C.
MALTEPE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

AKILLI ETMENLER VE AKILLI ETMEN YÖNELİMLİ PROGRAMLAMA YAKLAŞIMI

Kadir ÇAMOĞLU

Yüksek Lisans Tezi

Tez Danışmanı
Prof. Dr. E. Murat ESİN

İSTANBUL – 2010

T.C.
MALTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

AKILLI ETMENLER VE AKILLI ETMEN YÖNELİMLİ PROGRAMLAMA YAKLAŞIMI

YÜKSEK LİSANS TEZİ

Kadir ÇAMOĞLU

Tez Danışmanı
Prof. Dr. E. Murat ESİN

İSTANBUL – 2010

Bu tez çalışması, Maltepe Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... / ..... / ..... tarih ve ..... / ..... sayılı kararıyla oluşturulan jüri tarafından ***Bilgisayar Mühendisliği Yüksek Lisansı Tezi*** olarak kabul edilmiştir.

JÜRİ

Prof. Dr. E. Murat ESİN

Danışman

Yrd. Doç. Dr. Oruç Raif ÖNVURAL

Üye

Yrd. Doç. Dr. Birim BALCI

Üye

ÖZET

Yüksek Lisans Tezi, Akıllı Etmenler ve Etmene Yönelik Programlama Yaklaşımı, T.C. Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı.

Etmene Yönelik Programlama (EYP), Nesneye Yönelik Programlama (NYP) yöntemi gibi bir programlama yaklaşımıdır. EYP, pasif olan "nesne" yapısını baz alan NYP'yi geliştirerek aktif davranışlar sergileyen ve mental özellikleri bulunan "etmen" kavramı üzerine inşa edilmiştir. Akıllı yazılım etmenleri olarak da ifade edilen bu birimler, fikir, taahhüt, niyet, karar verme, plan seçme gibi mental kabiliyetlere sahip özerk varlıklardır.

Yapay zekâ çalışmalarının bir uzantısı olarak gelişen EYP, günümüzün karmaşık yazılım problemlerinin çözümünde güçlü bir alternatif olarak ortaya çıkmaktadır. Gerçek yaşamdaki aktif nesneleri temsil eden etmenlerle NYP ile gerçekleştirilmesi zor olan işleri başarmak kolaylaşmaktadır.

Bu tezin amacı, yapay zekâ hakkında genel bir görüş oluşturarak etmen kavramını tanımlamak, etmene yönelik programlamayı açıklamak ve etmene yönelik programlama ile nesneye yönelik programlama yaklaşımları karşılaştırarak daha iyi bir kavrayış sağlamaktır. Tez ile birlikte sunulan örnek proje bir çok-etmenli sistemin modellenmesini ve gerçekleştirilmesini içermektedir.

Bu tez 2010 yılında tamamlanmıştır ve 76 sayfadan oluşmaktadır.

Anahtar Kelimeler: Etmene yönelik programlama, akıllı etmenler, yapay zekâ, nesne yönelimli programlama.

ABSTRACT

Master Thesis, Intelligent Agents and Agent Oriented Programming, T.C. Maltepe University, Graduate School of Natural and Applied Sciences, Department of Computer Engineering.

Similar to Object-Oriented Programming (OOP), Agent-Oriented Programming (AOP) is a programming paradigm. AOP is built on the concept of an agent that has mental capabilities and exhibits active behavior. This is an improvement over OOP which takes a passive object structure as a base. These units which can also be referred to as intelligent software agents are the autonomous beings having beliefs, commitments, intensions, decision making and plan selection.

AOP, which is developing as an extension of studies related to artificial intelligence is emerging as a strong alternative in solving contemporary complicated software programming problems. It becomes easier to accomplish the tasks that are difficult to put into practice using OOP, if we use agents that represent proactive real life objects.

The objective of this thesis is to describe the concepts of agents creating a general view about artificial intelligence, and to explain Agent Oriented Programming. We also compared the AOP and OOP approaches with a view of their use in the development of pro-active solutions. The sample project presented with the thesis contains a model and realization of a multi-agent system.

This thesis is completed in 2010 and consists of 76 pages.

Keywords: Intelligent agent, agent oriented programming, artificial intelligence, object oriented programming

TEŞEKKÜR

Yüksek lisans eğitimim konusundaki destek ve yönlendirmelerinden dolayı Maltepe Üniversitesi rektörü sayın Prof. Dr. Kemal KÖYMEN'e, bu tez konusunu seçmem için beni yönlendiren ve tez süreci boyunca destek, yardım ve yönlendirmeleriyle değerli bilgi ve birikimlerinden faydalandığım danışman hocam Sayın Prof. Dr. E. Murat ESİN'e, çalışma boyunca fikirleri, bilgileri ve kaynakları bana destek olan hocalarım Sayın Yrd. Doç. Dr. Raif ÖNVURAL'a, Sayın Öğr. Gör. Fatih YÜCALAR'a, Sayın Öğr. Gör. Erdal GÜVENOĞLU'na ve Sayın Ar. Gör. Selim BAYRAKLI'ya, bu süreçte beni sabır ve anlayışla destekledikleri için sevgili aileme ve bu tezi oluşturacak bilgi ve birikimi oluşturmamda emeği geçen herkese teşekkürlerimi sunarım.

İÇINDEKILER

KISALTMALAR

İng. Kısaltma	İngilizce	Açıklama
FIPA	The Foundation for Intelligent Physical Agents	Akıllı Fiziksel Etmenler için Temeller
OMG	Object Management Group	Nesne Yönetim Grubu
ACL	Agent Communication Language	Etmen İletişim Dili
MAS	Multi-Agent Systems	Çoklu etmen sistemleri
MMAS	Mobile Multi-Agent Systems	Mobil çoklu etmen sistemleri
BDI	Belief – Desire – Intention	Fikir, İstek, Niyet
IDE	Integrated Development Environment	Bileşik Geliştirme Ortamı
AOSE	Agent Oriented Software Engineering	Etmene Yönelik Yazılım Mühendisliği
OOP	Object Oriented Programming	Nesneye Yönelik Programlama
AOP	Agent Oriented Programming	Etmene Yönelik Programlama

TERIMLER

Türkçe	**İngilizce**	**Açıklama**
Yazılım Etmeni	Software Agent.	Belirli bir donanıma sahip olmaksızın, bir bilgisayar ya da gömülü sistem üzerinde çalışan, sürekli çalışır konumda olan ve etmen özellikleri gösteren yazılım.
Çok etmenli sistemler	Multi-agents systems	
Tutum.	Behavior	Nesnelerin davranış ve tepkilerin toplamıdır.
Fikir	Beleif	Etmenin sahip olduğu durum bilgisi, o etmenin inancını oluşturur.
Niyet	Intention	Etmenin belirli planları uygulayarak amacına ulaşmaya çalışmasıdır.
Özerk	Autonomous	Etmenler içlerinde bulundukları ortamda başka bir şeye bağlı olmaksızın tamamen kendi başlarına var olabilir ve eylemde bulunabilirler
Proaktiflik	pro-activeness	Etmenler dışarıdan bir etki gelmeksizin kendi başlarına harekete geçebilirler.
Dayanıklılık	robustness	Bir etmenin herhangi bir nedenle çalışması kesilirse, çalışmaya yeniden başladığında kararlı bir durumdan amacına ulaşmak için çalışmaya devam edebilmelidir.

ŞEKILLER

1. GİRİŞ

Bu çalışmanın konusunu yazılım mühendisliğinde son dönemde öne çıkan yazılım geliştirme paradigmalarından biri olan "Etmene Yönelik Programlama" yaklaşımı oluşturmaktadır. Yazılım geliştirme paradigmaları, bir problem için yazılım geliştirirken, problemin çözümüne hangi bakış açısıyla yaklaşılması gerektiğini belirleyen metotlardan oluşur. Yazılım geliştirme paradigmaları çözümü oluşturacak olan kodun nasıl paketleneceğini (nesne, fonksiyon, modül, bileşen, etmen, vb.), paketlerin kendi içlerinde hangi yapıda olacağını ve birbirleriyle nasıl entegre olarak çözümün tamamını oluşturacaklarını tarif eder. Yazılım geliştirme paradigmaları sadece nelerin yapılması gerektiğini tarif etmekle kalmazlar aynı zamanda nelerin yapılmaması gerektiğini de söylerler. Böylece tercih edilen yaklaşım kullanıldığında karmaşık problemlere daha kolay çözüm üretilmesi hedeflenir.

Günümüzde iş dünyasında ve günlük yaşamda en yoğun kullanılan programlama paradigmaları nesne yönelimli programlama - NYP (object oriented programming) , fonksiyonel programlama (functional programming), cepheye yönelik programlama (aspect oriented programming), mantıksal programlama (logic programming) ve etmene yönelik programlamadır- EYP (agent oriented programming). Farklı yazılım problemleri için yukarıdaki farklı paradigmalardan uygun olanlarını kullanmak, hem çözüm geliştirme sürecini verimli kılacak hem de geliştirilen çözümün başarımını artıracaktır.

1980'li yıllardan bu yana iş dünyası uygulamalarında ağırlıklı olarak kullanılan NYP, problem alanını gerçek yaşamda olduğu gibi nesnelere ve bu nesneler arasındaki ilişkilere parçalayarak paketler ve bu şekilde en uygun çözümü bulmaya çalışır. Çoğu durumda çözüm geliştirme konusunda yeterli olan bu yaklaşım, iş dünyasındaki dinamik değişimler nedeniyle son zamanlarda ortaya çıkan bazı ihtiyaçlara tek başına cevap vermekte zorlanmaya başlamaktadır. Kullanıcı yerine hareket edebilecek, sistemde bir kere var olduktan sonra sürekli olarak varlığını devam ettirecek, belirli amaçları olan ve bu amaçlar doğrultusunda kendisine tanımlanmış belirli planları uygulayan, etraflarını algılayarak hareketlerini buna göre şekillendirecek uygulamalara olan ihtiyaç, akıllı etmenleri ve etmene yönelik programlamayı ortaya çıkartmıştır.

Bu çalışmanın konusu akıllı etmenleri ve EYP'yı açıklamak ve örneklendirmektir. Çalışmanın ikinci bölümünde Yapay Zekâ başlığı altında akıllı etmenlerin ve EYP'nın ortaya çıkışını sağlayan yapay zekâ çalışmalarından bahsedilmektedir. Burada yapay zekânın tarihinden kısaca bahsedilerek, temel yapay zekâ çalışma alanlarına değinilmiştir. Ardından akıllı etmenler, etmenlerin temel nitelikleri, karakteristik özellikleri, türleri ve kullanım alanlarıyla birlikte açıklanmıştır. Sonrasında etmene yönelik programlama konusuna geçilmiş ve bu başlık altında etmene yönelik yazılım geliştirme döngüsü, etmen geliştirme dil, metodoloji, ortam ve altyapıları ve standartlar anlatılmıştır. Ardından NYP hakkında bilgi verilmiş ve EYP ile NYP karşılaştırılmıştır. Çalışmanın sonunda etmene yönelik bir uygulama örneğinin tasarlanması ve geliştirilmesi anlatılarak konunun pekiştirilmesine çalışılmıştır.

2. YAPAY ZEKÂ

Zekâ, insanın çevresindeki gerçekleri algılama, değerlendirme, yargılama ve belirli bir sonuca ulaşma yeteneklerinin tamamıdır. Her insan doğuştan belirli bir zekâ düzeyine sahiptir. Ancak bilinçli bir çalışma, eğitim, birikim ve deneyimle zekâ düzeyi artırılabilir. Daha önce karşılaşılmamış bir duruma uyum sağlayabilme, durumu kavrama, analiz yeteneği, duyuların, dikkat ve düşüncenin bu olaya yoğunlaştırılabilmesi ve sonuçta bu durumdan yeni bir deneyim edinme zekâ ile gerçekleştirilmektedir. [1]

2.1. Yapay Zekânın Kısa Tarihçesi

İnsanın kendine benzer zeki sistemler geliştirme çabası, altından yapılmış robot yaratıklardan bahsedilen antik Yunan mitolojisine kadar dayanmaktadır. İlk çalışan otomatik sistemlerle ilgili bilgilere ise antik Yunan ve Mısır tarihinde rastlanılmıştır.[2]

Gerçek anlamda yapay zekâyla ilgili çalışmalar 1940'lı yılların başında elektronik bilgisayarın icadıyla başlar. Öncesinde de yapay zekâya olarak nitelendirilebilecek birçok çalışma olmasına rağmen, yapay zekânın vizyonunu tam olarak ortaya koyan isim olarak 1950 yılında yayınlanan "Computing Machinery and Intelligence" isimli makalesiyle Alan Turing'dir. Turing bu çalışmasında makine öğrenmesi, genetik algoritma, destekli öğrenme ve Turing testlerinden bahsetmişti. [3]

"Yapay zekâ" kavramı ise, 1956 yazında Dartmouth kolejinde otomata teorisi, zekâ ve sinir ağlarıyla ilgili 10 kişilik bir Amerikalı araştırmacı grubunun yaptığı iki haftalık çalışma sonrasında, John McCarthy tarafından ortaya atılmış ve ayrı bir çalışma alanı olarak teklif edilmiştir.[3]

2.2. Yapay Zekânın Tanımı

Yapay zekâyla ilgili birçok farklı araştırmacı birçok tanım yapmıştır. Bu konuda tüm bu tanımları genel hatlarıyla özetleyen bir tanım Vasif Nabiyev tarafından şöylece ifade edilmiştir: "Yapay zekâ, kabaca; bir bilgisayarın ya da bilgisayar denetimli bir makinenin, genellikle insana özgü nitelikler olduğu varsayılan akıl yürütme, anlam çıkartma, genelleme ve geçmiş

deneyimlerden öğrenme gibi yüksek zihinsel süreçlere ilişkin görevleri yerine getirme yeteneği olarak tanımlanmaktadır.” [4]

Yapay zekânın amacı insan zekâsına sahip bilgisayarları geliştirmek, insanın zeki davranışlarıyla benzeşen makineler yapmaktır. Yapay zekâ bunu yaparken bilgisayar mühendisliği bilimini esas almakla birlikte matematik, ekonomi, sinirbilim, psikoloji, kontrol teorisi, sibernetik ve dilbilimi alanlarıyla da birlikte çalışır. [3,5]

Yapay zekâ temel olarak

- İnsan gibi düşünen
- İnsan gibi davranan
- Akılcı düşünen
- Akılcı davranan

sistemlerle uğraşır. Bu çabaların ortak yönü, bilginin akılcı ve insanda olduğu gibi işlenmesinin ilkelerini araştırıp bulmaktır. [3,5]

2.3. Yapay Zekânın Çalışma Alanları

Yapay zekâ araştırmacıları bu çabaları ilk başta aşağıdaki dört temel alan üzerinde yoğunlaştırmışlardır:

- Doğal dil işleme sistemleri
- Görüntü işleme ve dönüştürme sistemleri
- Robotik
- Uzman sistemler

Zamanla insan zekâsını yapay ortamlarda gerçekleştirme çabasıyla yeni çalışma alanları açılmaya ve teknolojiler gelişmeye başladı. Bunlardan bazıları aşağıdaki gibidir: [3,5,6]

- Açıklama tabanlı öğrenme
- Algılayıcılar
- Anlama
- Benzerliğe dayalı öğrenme

- Bilgi Tabanlı Sistemler (Bilgi gösterimi, uzman sistemler, simülasyonlar)
- Bilimsel buluşların modellenmesi
- Çıkarım veya sağduyu bilgi işleme
- Dağıtılmış yapay zekâ
- Doğal dil işleme
- Endüktif öğrenme
- Genetik algoritmalar
- Geometrik muhakeme
- Görme/görüntü işleme
- Kaos teorisi
- Kavramsal grafikler
- Konuşma
- Makine Buluşları (Bilgi Madenciliği, Bilimsel buluşların Modellendirilmesi)
- Makine Öğrenmesi (Bilgi düzeyinde öğrenme, sembol düzeyinde öğrenme, aygıt düzeyinde öğrenme)
- Mantık programlama
- Model tabanlı muhakeme
- Monotik olmayan muhakeme veya doğruyu koruma mekanizması
- Nesne tabanlı zeki sistemler
- Oyunlar (satranç, dama, strateji, vb.)
- Paralel yapay zekâ sistemleri
- Planlama, tabloluma, zaman çizelgeleri oluşturma
- Problem çözme, arama ve sezgisel programlama
- Robotik
- Sekil Tanıma (Nesne tanıma, Optik Karakter Tanıma (OCR), Ses Tanıma)
- Teorem ispatlama
- Uzman sistemler
- Veri tabanlı muhakeme
- Yapay sinir ağları
- Yapay yaşam (hayat)
- Zeki etmenler
- Zeki veritabanları

3. AKILLI ETMENLER

Yapay zekâ alanındaki çalışmalarda, geliştirilen sistemlerin tümü başlangıçta "yapay zekâ sistemi" olarak adlandırılırken, zamanla bu sistemlerden kendi başlarına özerk olarak ayrılanları "etmen" olarak isimlendirilmeye başlanmıştır. Özellikle yapay zekânın "Dağıtık Yapay Zekâ" başlığı üzerinde çalışma yapanlar, konuyu "Dağıtık Problem Çözme ve Çok-Etmenli Sistemler" olarak iki ayrı kola ayırmışlardır. [7]

Böylece belirli bir amacı olan, etrafını algılayan, algıladıklarını anlamlandıran ve bu anlamlandırma neticesinde çeşitli tepkiler veren özerk yapay zekâ sistemlerine, "etmen" denilmeye başlanmıştır.

Etmenler gerçek yaşamı modeller; gerçek yaşamdaki kendi hedefleri olan, birbirleriyle iletişim kuran ve ortak çıkarları için çalışan birimleri temsil ederler. Bu bakış açısıyla etmenlerin nesneye yönelik programlamayı genişlettiği söylenebilir. Bununla birlikte etmen teknolojisi Nesneye Yönelik Programlama, Dağıtık Hesaplama, Paralel Hesaplama, Mobil Programlama gibi bir grup önemli bilgisayar teknolojisinin bir araya gelmesiyle oluşur. [8]

Yapay zekânın bir alt çalışma alanı olarak başlayan etmene yönelik çalışmalar, zamanla hem yapay zekânın diğer alt alanlarıyla ilişkisi hem de bilgisayar bilimleri ve özellikle de yazılımla sıkı ilişkisiyle devam etmektedir. "Kendi başına çalışabilen sistem" teması üzerine kurulu olan "etmen" kavramı, gelişen bilgisayar teknolojileri ve yazılım mühendisliği sayesinde gittikçe gelişmektedir.

Bu başlıkta akıllı etmenlerin genel özellikleri ve temel bileşenleri açıklanmaktadır.

3.1. Akıllı Etmenin Tanımı

Etmen (agent) ve akıllı etmen (intelligent agent) kavramları üzerine birçok çalışma olmasına rağmen, halen herkesçe üzerinde anlaşılmış genel bir tanım bulunmamaktadır. Hatta bazı çalışmalarda bu iki kavramın birbirlerinin yerine kullanıldığı gözlenmiş olsa da etmen daha geniş bir kavram olarak karşımıza çıkar. Akıllı etmen ise etmen kavramının bir alt türüdür.

Shoham'a göre "bir etmen, fikirleri (beliefs), yetenekleri (capabilities), seçimleri (choices) ve taahhütleri (commitments) gibi mental özellikleriyle belirli bir görüşe sahip olan bir varlıktır."[9]

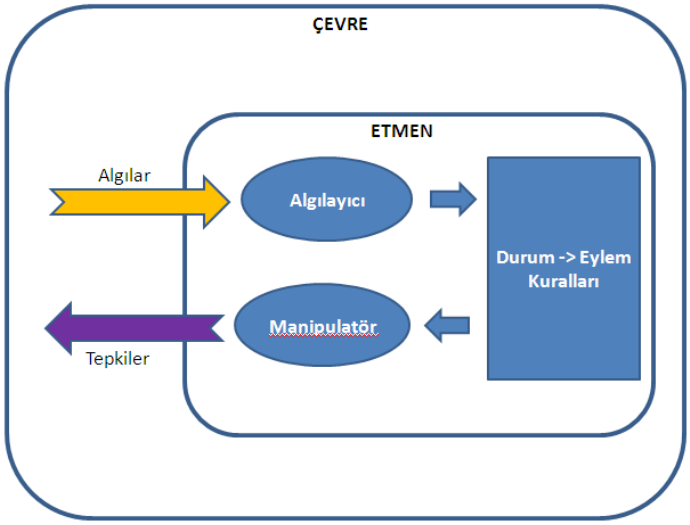
Wooldridge' e göre akıllı etmenin tanımı şöyledir: "Akıllı etmen, belirli bir ortamda konumlanmış bir bilgisayar sistemidir ve tasarımında belirlenmiş olan amaçlara ulaşmak için özerk olarak davranışlarda bulunma kabiliyetine sahiptir."[10]

Sandeep' e göreyse "bir etmen, otonomi, sosyallik, reaktiflik ve pro-aktiflik özelliklerine sahip bilgisayar tabanlı yazılım ya da donanım" olarak belirtilmiştir.[11]

Maes'e etmenleri tanımlarken "otonom etmenler" ifadesini kullanmıştır. Maes'e göre; "otonom etmenler karmaşık ve dinamik bir ortamda yaşayan ve bu ortam içerisinde otonom olarak çevresini algılayan ve eylem gerçekleştiren ve böylece yapmak için tasarlandıkları amaçları ya da görevleri gerçekleştiren sayısal sistemlerdir."[12]

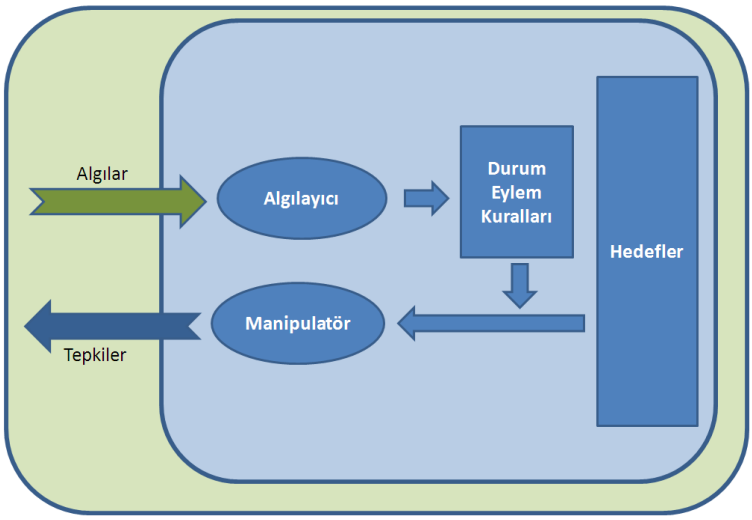
Hayes ve Roth ise "akıllı etmen" deyimini kullanmış ve tanımlarını şöyle yapmışlardır: "Akıllı etmenler devamlı olarak üç fonksiyon gerçekleştirirler: içlerinde bulundukları ortamın dinamik durumlarını algılarlar; ortamı etkilemek üzere bir takım eylemlerde bulunurlar ve algıladıklarını muhakeme edip problem çözer, sonuç çıkartır ve hangi eylemleri gerçekleştireceklerini belirler."[13]

Basit bir tepkisel etmen aşağıdaki şekil 3.1 deki gibi bir görselle ifade edilebilmektedir.



Şekil 3 1 Basit Tepkisel Etmen

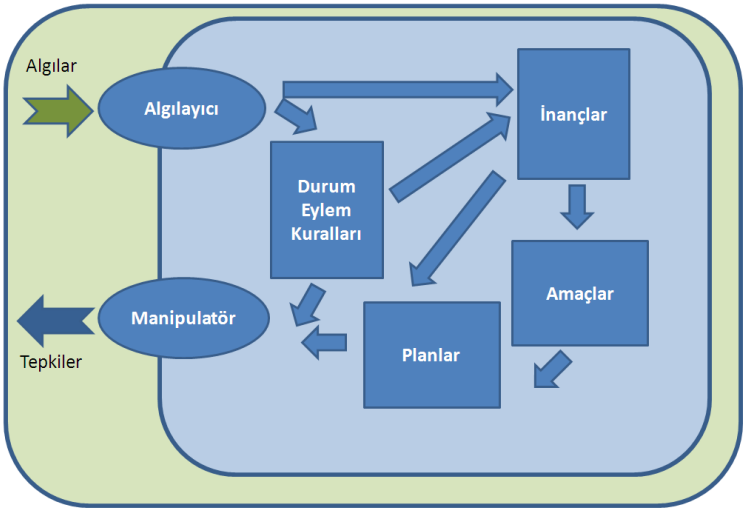
Şekil 3.1'deki basit tepkisel etmen belirli algılayıcıları sayesinde içinde bulunduğu ortamdaki değişiklikleri algılar ve daha önceden belirlenmiş durum/eylem kurallarına bağlı olarak manipülatörleri aracılığıyla çevresinde bir değişim yaratır. Örneğin termostatlı bir ısıtıcı basit bir refleks etmenidir.[14] Çevresindeki sıcaklık değişimlerini algılar ve ortamın ısısı ayarlananın altına düştüğünde ısıtıcı çalışır. Ortam yeterince ısındığında termostat bu sıcaklık artışını tespit eder ve ısıtıcının çalışması durur.



Şekil 3 2 Proaktif Etmen

Yukarıdaki şekil 3.2 de temel yapısı gösterilen reaktif etmen, tepkisel etmenden farklı olarak belirli hedeflere sahiptir ve sadece dışarıdan gelen değişikliklere tepki vermekle kalmaz, ayrıca proaktif olarak belirli işler de yapar. Örneğin bir noktadan diğerine gitmeye programlanmış olan bir robot, yolda karşısına çıkan engeller karşısında ne yapacağını tepkisel olarak belirlerken, proaktif oluşu nedeniyle bu engeli aştıktan sonra hedefine gitmek üzere yeniden yola koyulur ve hedefine ulaşıncaya kadar da yoluna devam eder.

Aşağıdaki Şekil 3.3 deki daha gelişmiş FİN (fikir – istek – niyet) etmense, yine algılayıcılarıyla etrafındaki değişiklikleri izler. Değişikliklerle ilgili edindiği bilgileri bir taraftan biriktirirken, bir taraftan da değişen duruma gerekli tepkiyi verir. Ayrıca kendi amaçları için proaktif olarak hareket eder. Proaktif etmene ek olarak, amacına ulaşmak için birden fazla plana sahiptir ve değişen durumlar karşısında değişik planları uygulayarak amacına ulaşmaya çalışır.



Şekil 3 3 FİN Etmen

Etmenlere örnek olarak en ileri, yüksek uç olarak insan verilebilir. İnsan birçok amacı, algılayıcısı, manipülatörü ile en ileri etmen örneğidir. Beş duyusuyla etrafını sürekli olarak algılar ve değişimlere konuşarak, hareket ederek tepki verir. Ayrıca belirli amaçları için, herhangi bir etki olmaksızın çevresindeki mevcut durumu baz alarak kendisi bir hareket başlatabilir.

3.2. Akıllı Etmenlerin Karakteristik Özellikleri

Etmenlerin tanımında genel bir kanıya varılamaması aslında etmenlerin karakteristik özellikleri ve sınıflandırılmasıyla da ilgili farklı görüşlerin ortaya atılmasıyla ilgilidir. Birçok araştırmacı hem tanım hem de karakteristik özellikler konusunda farklı düşünceler ileri sürse de hemen hemen herkesin üzerinde uzlaştığı temel öğeler şöyledir.

3.2.1. Fikir (belief)

Etmenlerin içinde bulundukları ortamla ve kendi durumlarıyla ilgili mevcut bilgisini ifade eder. Bu bilgi, etmenin algılayıcıları yoluyla ve diğer etmenlerle iletişimi sayesinde aldığı bilgilerden oluşur.

3.2.2. Amaç (desire):

Bu karakteristik özellik, etmenin ulaşmak istediği noktaya geldiğinde oluşacak durumu ifade eder. Örneğin havaalanı üzerinde daire çizen bir uçağın yere güvenli bir şekilde inmesi, uçağın amacıdır.

3.2.3. Niyet (intention):

Niyet, etmenin kendi yapısında daha önceden tanımlanmış belirli planları uygulanarak, amacına ulaşmaya çalışmasıdır.

3.2.4. Amaca yönelik olmak (goal-oriented):

Etmenler sadece içlerindeki bulundukları ortamdaki değişikliklere tepki göstermekle kalmazlar, ayrıca belirli bir amaca ulaşmak için gerekli eylemleri belirli bir dış etmene bağlı ya da bağımsız olarak gerçekleştirirler.

3.2.5. Özerklik(autonomous):

Etmenler insan ya da başka bir varlığın müdahalesi olmaksızın kendi başlarına hareket edebilirler (davranış sergileyebilirler). Durumlarını ve tepkilerini kendileri belirleyebilirler.

3.2.6. Sosyallik(social-ability):

Etmenler çevreleriyle, diğer etmenlerle ve varoluş sebeplerine bağlı olarak belirli birimlerle iletişimde bulunurlar.

3.2.7. Reaktiflik(reactivity):

Etmenler çeşitli algılayıcılarla içinde bulundukları ortamdaki değişiklikleri algılarlar ve bu algılamaya bağlı olarak fikirlerini değiştirebilir ve zamanında tepki göstermek üzere bir aksiyonda bulunurlar.

3.2.8. Proaktiflik (pro-activeness)

Etmenler sadece etraflarında olan olaylara ve değişimlere tepki göstermekle kalmazlar ayrıca amaçlarını gerçekleştirmek için dışarıdan bir etki gelmeksizin amaçlarını gerçekleştirmek için harekete geçerler.

3.2.9. Hareketlilik(mobility)

Bazı etmenler (mobil) içlerinde bulundukları ortamda hareket edebilirler. Bazı etmenler belirli bir bilgisayarda başlayan bir işlemi diğer bir bilgisayara giderek devam ettirebilirler.

3.2.10. Dürüstlük(veracity)

Bir etmenin bilerek, kasıtlı olarak yanlış bilgi vermeyeceği varsayımıdır. Bu bir etmenin diğerinden aldığı bilginin doğru olacağı ön kabulünü ifade eder.

3.2.11. Tabiilik (benevolence)

Etmenlerin birbirleriyle çelişen hedeflere sahip olamayacağı ve bir etmenin kendisine verilmiş olan görevi yerine getirmeye adanacağı varsayımıdır.

3.2.12. Rasyonellik (rationality):

Bir etmenin tamamen kendi amaçlarına yönelik eylemler yapacağı, amacına ulaşmayı engelleyecek eylemler gerçekleştirmeyeceği varsayımıdır.

3.2.13. Konumlanmışlık (situated):

Bir etmen belirli bir ortamda bulunur. Bir ortamın parçasıdır ve ortam içinde yer alır.

3.2.14. Esneklik(flexible):

Etmenler hedeflerine ulaşmak için birden fazla alternatif plana sahiptir. Etmen, bu planlardan duruma en uygun olanlardan herhangi birini seçme iradesine sahiptir.

3.2.15. Dayanıklılık (robustness):

Herhangi bir başarısızlık ya da ters giden durum karşısında görevine devam edebilir. Örneğin bir noktadan diğerine gitmek üzere programlanmış bir robot, yoluna çıkan bir engele takılarak devrildiğinde, kendini düzeltebilmeli ve düzeldikten sonra yeniden amacını gerçekleştirmeye kaldığı yerden devam edebilmelidir.

3.2.16. Süreklilik/kesintisizlik (continuous):

Etmenler içlerinde bulundukları ortamda sürekli, kesintisiz olarak, çalışırlar. Bu çalışma ortam var olduğu sürece devam eder. Herhangi bir sebeple çalışması engellenen, bozulan bir etmen kendini yeniden ayağa kaldırabilmeli ve tutarlı bir noktadan yeniden başlayarak amacına yönelebilmelidir.

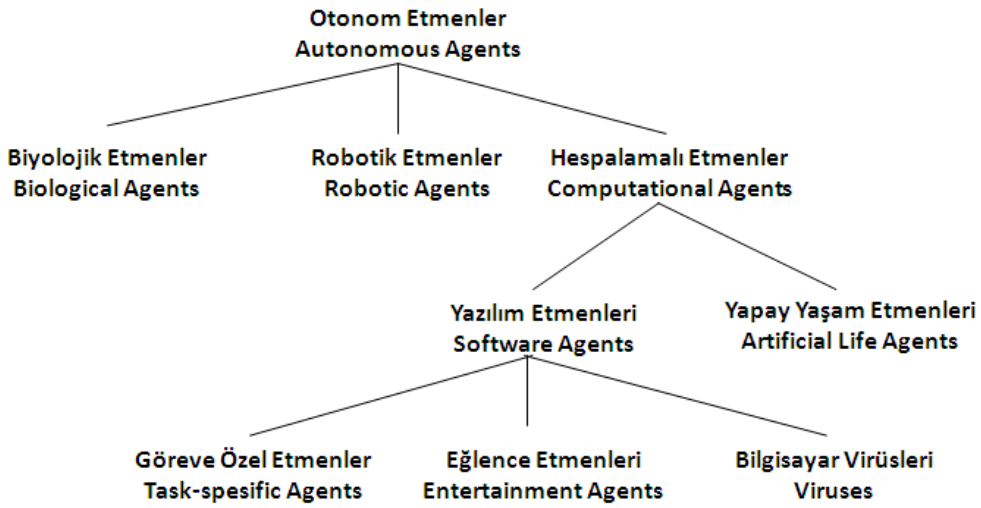
3.2.17. Uyum sağlayan (learning/adaptive):

Önceki deneyimlerine dayanarak fikirlerini ve hareketlerini değiştirirler.

3.3. Akıllı Etmenlerin Sınıflandırılması

Etmen tanımıyla ilgili herkesçe kabul edilmiş ortak bir görüş olmadığı gibi, etmenlerin sınıflandırılmasıyla ilgili de ortak bir kanı yoktur. Birçok araştırmacı etmenleri farklı açılardan ele alarak çeşitli sınıflandırmalar yapmıştır.

Franklin ve Graesser tarafından yapılan etmen sınıflandırmasında başlangıç olarak otonom etmenler alınmıştır. Biyolojik, donanımsal ve yazılımsal etmenlerin tamamı otonom etmen olarak kabul edilmiştir. Aşağıdaki şeklide Franklin ve Graesser tarafından yapılan sınıflandırmayı görmektesiniz.[14]



Şekil 3 4 Etmenlerin Sınıflandırılması

Biyolojik, robotik ve hesaplamaya dayalı etmenler olarak gelen ikinci seviyede hesaplamaya dayalı etmenler de kendi içlerinde yazılım etmenleri ve yapay yaşam etmenleri olarak ikiye ayrılır. Bu çalışmanın konusunu oluşturan yazılımsal etmenler için Franklin ve Graesser üç alt sınıf belirlemiştir.

- Göreve özel etmenler
- Eğlence etmenleri
- Bilgisayar virüsleri

Russell & Norvig ise "Artificial Intelligence: A Modern Approach" isimli kitaplarında etmenleri beş sınıfa ayırmıştır.[3]

1. Basit tepki etmenleri (simple reflex agents)
2. Modele dayalı tepki etmenleri (model-based reflex agents)
3. Amaca dayalı etmenler (goal-based agents)
4. Faydaya dayalı etmenler (utility-based agents)
5. Öğrenen etmenler (learning agents)

Etmenler diğer bir açıdan gerçekleştirdikleri iş ve işlemlere göre de sınıflandırılabilirler.[15]

1. Karar Etmenleri (Decision Agents): Karar verme işlemleri için kullanılırlar.
2. Veri Giriş Etmenleri (Input Agents): Algılayıcılarla içlerinde bulundukları ortamı algılar ve gelen verileri işlerler.
3. Yürütme Etmenleri (Processing Agents): Belirli bir problemin çözümüne odaklanmış etmenlerdir. Örneğin ses tanımlama/algılama, görsel çözümleme, vb.
4. Uzamsal Etmenler (Spatial Agents): Fiziksel gerçek dünyayla ilgili etmenlerdir.
5. Evren Etmenler (World Agents): Diğer sınıftan otonom etmenlerin birlikte çalışmalarını sağlar.
6. Fiziksel Etmenler(Physical Agents): Algılayıcılarla içlerinde bulundukları ortamı algılayıp, çeşitli bileşenleriyle tepki veren, hareket sergileyen donanımsal/fiziksel birimlerdir.
7. Zamansal Etmenler (Temporal Agents): Zamana bağlı olarak kayıtlı çeşitli bilgileri kullanarak ve etrafını algılayarak çeşitli tepkiler verir ve işlemler yapar.

Etmenler ayrıca çalıştıkları ortama ve çalışma şekillerine bağlı olarak

- Özerk (autonomous),
- mobil (mobile)
- çok etmenli (multi-agent)
- ve dağıtık (distributed)

etmenler olmak üzere dörde ayırmak da mümkündür. Her etmen doğal olarak kendi başına çalışabilir ve özerk olmalıdır. Aksi takdirde o akıllı sistemin bir "etmen" olması söz konusu olamaz. Mobil etmenler belirli bir ortamdan diğerine geçebilen özerk etmenlerdir. Mobil etmenler belirli bir ortamda çalışıyorlarken, durumlarını ve yaptıkları işleri kendi belirledikleri bir ortama yine kendi belirledikleri bir zamanda taşıyarak çalışmalarına devam ederler. Dağıtık etmenlerse, belirli bir işi farklı ortamlardaki farklı kaynakları kullanarak daha kısa sürede ve verimli şekilde gerçekleştirmek için geliştirilmiş birden fazla etmenden oluşurlar.

3.4. Akıllı Etmenlerin Uygulama Alanları

Akıllı etmenler yazılım geliştirmede yeni bir paradigma olarak hızla yaygınlaşmaktadır. Başlangıçta yapay zekânın alt çalışma alanlarından bir olarak karşımıza çıkan etmenler, zamanla günlük hayattaki uygulamalarda da yerlerini almaya başlamışlardır. E-posta filtrelemeden trafik yönetim sistemlerine kadar çok çeşitli alanlarda kullanılmaya başlanan etmenlerin kullanım alanlarına sürekli yenileri eklenmektedir. [16]

Jennings ve Wooldridge'e göre akıllı etmenlerin uygulama alanlarını aşağıdaki gibi sınıflandırmak mümkündür [16]:

1. Endüstriyel Uygulamalar
   1.1. Süreç Kontrolü
   1.2. Üretim
   1.3. Hava Trafik Kontrolü
2. Ticari Uygulamalar
   2.1. Bilgi Yönetimi
   2.2. Elektronik Ticaret
   2.3. İş Süreçleri Yönetimi
3. Tıbbi Uygulamalar
   3.1. Hasta İzleme
   3.2. Sağlık Hizmetleri
4. Eğlence
   4.1. Oyunlar
   4.2. Etkileşimli Sinema ve Tiyatro

Pour ise etmene yönelik yazılım geliştirmeyi yazılım mühendisliği müfredatı için uyarladığı çalışmasında etmenlerin uygulama alanlarını aşağıdaki gibi listeler. [17]

- Ağ işlemleri
- E-Ticaret
- Bilgi Yönetimi (sağlık hizmetleri, eğitim, kayıp çocuk, otomobiller, yolculuk, emlak, çalışma ortakları, istihdam)

- Finans
- Bankalar (sanal bankacılık)
- Perakende mağazalar
- Medya
- Devlet (uzay keşfi)
- Bilgi teknolojileri
- Üretim
- Kişisel yardımcı
- Giyilebilir bilgisayarlar
- Akıllı elbiseler
- Sanal akvaryum
- Sanal ev hayvanları

3.5. Akıllı Etmen Uygulamaları Geliştirmeyle İlgili Sıkıntılar ve Tartışmalı Konular

Akıllı etmenler, her türlü yazılımsal sorunu çözmekte kullanılabilecek tek bir araç olarak düşünülmemelidir. Akıllı etmenlerle programlama, önceki programlama teknoloji ve yaklaşımları çözülemeyen birçok soruna yeni ve etkin çözüm getirmiş olmakla birlikte, birçok yazılım probleminin eski teknik ve yöntemlerle çözülmesi daha verimli olduğu da herkesçe kabul edilmektedir. Bununla birlikte etmenlerin yaygınlaşması ve daha verimli kullanılması ancak etmene yönelik geliştirilen yazılımların başarılarıyla mümkün olacaktır. Bu noktada akıllı etmen uygulamalarının geliştirilmesindeki sıkıntılar ve tartışmalı konular dikkatle incelenmeli ve burada belirtilen tuzaklara düşülmemeli, teknolojinin henüz yetersiz kaldığı alanlar için gerekli çözümler bulunmalıdır.

Wooldridge ve Jennigs etmene yönelik uygulama geliştirmenin potansiyel sıkıntılarını altı grup altında değerlendirmişlerdir: [18]

1. politik,
2. yönetimsel,
3. kavramsal,
4. analiz/tasarım,
5. etmen seviyesi,
6. dağıtık seviye.

Bazı gerçek yaşam problemleri, etmen temelli olmayan yazılım geliştirme yaklaşımlarıyla çözülemezken, etmene yönelik sistemlerle çözülebilecektir. Ancak buna karşı dogmatik bir direncin yaşanması söz konusu olacaktır.

Yönetimsel sıkıntılar ise, daha çok etmen tabanlı uygulamalardan ne isteyeceğini tam olarak bilmeyen yöneticilerin, etmene yönelik projeler başlatması ve başlangıçta projeden ne istendiğinin bilinmeyişi nedeniyle de projelerinin başarısız olması ihtimalidir.

Kavramsal sıkıntılarla ifade edilmek istenen, yazılımcıların etmenlerin birer yazılım olduğunu unutarak, etmenlerden çok şey beklemesi ve her türlü sorunu çözmesini ummasıdır. Eğer etmen kavramına bu şekilde "her şeyi çözebilecek bir şey" gözüyle bakılırsa, daha başından başarısızlığa mahkûm edilmiş projeler ciddi bir sıkıntı oluşturacaktır.

Yazılımcılar analiz ve tasarım aşamasında yapacakları işlerle ilgili teknolojileri pas geçer, yazılım mühendisliği metodolojilerini uygulamazlarsa, etmen tabanlı uygulama geliştirme sürecinde sorun yaşayacaklardır.

Eğer yazılımcılar etmen geliştirirken çok fazla ya da çok az yapay zekâ kullanırlarsa, geliştirdikleri projede sıkıntı oluşacaktır.

Dağıtık seviyede oluşabilecek sorunlarsa, her yerde etmenlerin olması ya da çok az sayıda, yetersiz sayıda etmenin oluşturulması şeklinde özetlenebilir.

Diğer taraftan EURESCOM tarafından yapılan bir çalışmada, yazılım problemlerinin akıllı etmenlerle çözümü henüz yeterince olgunlaşmadığı için, etmen geliştirme üzerine bazı konuların henüz tartışma aşamasında olduğu tespit edilmiş ve bu konular yedi başlık halinde özetlenmiştir. [8]

1. Etmenlerin Kullanılabileceği Alanlar
2. Etmen Geliştirme Modelleri
3. Etmenlerin Test Edilmesi
4. Güvenlik ve Güven

5. Performans
6. Etik
7. İnsan – Etmen Etkileşimi

Etmen teknolojisi, tüm yazılımsal sorunlar için cevap olabilecek bir teknoloji değildir. Diğer teknolojilerin çözüm için daha uygun olduğu alanlar belirlenip, hangi durumlarda ve alanlarda etmenlerin kullanılması gerektiği üzerine çeşitli çalışmalar yapılmaktadır.

Her ne kadar etmen geliştirmek için kullanılabilecek birçok araç olsa da, bunların her birisi kendisi için özel bir etmen geliştirme yaklaşımı ve mimarisi sunar. Bu nedenle yaptığınız çalışma mecburen ürüne özel bir çalışma olarak kalacaktır. Etmen geliştirme araçlarının hepsiyle çalışacak bir modelin/mimarinin geliştirilmesi için zamana ihtiyaç vardır.

Etmenler yapıları gereği öğrenebilen ve adapte olabilen bileşenlerdir. Ayrıca etmenlerin içlerinde bulundukları durum ve ulaşmak istedikleri hedefler için birden fazla davranış gösterme yetenekleri olduğu için, test edilebilmeleri hayli zordur. Belirli bir hedef için belirli bir durum verildiğinde etmen farklı zamanlarda farklı davranışlar sergileyebilecektir. [8] Bu konuda yapılması gereken, etmenin amaçlarını gerçekleştirmeye yönelik hareketler sergileyip sergilemediğini test edecek sistemler geliştirmek olacaktır.

Etmen sistemler doğaları gereği temsil ettikleri istemcinin kişisel/özel bilgileriyle çalışacaklarından, bu bilgilerin güvenilir şekilde kullanılıyor olması gerekir. Özellikle mobil etmenler bilgileri farklı ortamlara taşıdıklarından bu konuda daha büyük bir tehdit olarak karşımıza çıkmaktadır. [8]

Etmen temelli bir sistemin performansı üzerinde çalıştığı sisteme, dağıtım ve iletişim alt yapısını bağlıdır. Ayrıca esnek etmenler, belirli bir göreve özel olarak geliştirilmiş etmenlerden doğal olarak daha yavaş çalışacaklardır. [8]

"Öğrenen ve öğrendiği bilgiler doğrultusunda farklı kararlar veren bir etmenin, aldığı bu kararlar doğrultusunda gerçekleştirdiği bir davranışın sorumlusu kim olacaktır?"[8] Topladığı ve filtreleyerek kullanıcısına sunduğu bir bilginin yanlış olması basit bir hata olsa bile bu bilgiye dayanarak yapılan işlemin sonuçları ciddi sorunlar doğurabilecektir. Diğer taraftan internet

üzerindeki etmenlerin sunuculara normal kullanıcıların yüzlerce belki binlerce fazlası istekte bulunması hem sunucu istatistiklerini yanlış yönlendirebilecek hem de kaynakların kullanımı açısından sıkıntı yaratabilecektir.

Etmenlerin varoluş sebebi, insanların yerine birtakım işlemleri gerçekleştirmektir. Burada bir etmenin yapması gereken, kullanıcısının/istemcisinin isteğini amaç olarak kabul ederek, bu amacı gerçekleştirmektir. Ancak bu noktada bir insanın duygusallığının da içinde bulunduğu karar alma sürecinin bir yazılım tarafından gerçekleştirilebilmesi, bir insanın ne istediğini tam olarak etmene nasıl aktaracağından başlayarak ciddi zorluklarla doludur. [8]

4. NESNEYE YÖNELİK PROGRAMLAMA

Nesneye Yönelik Programlama (NYP), günümüzde en yaygın kullanılan programlama paradigmasıdır. NYP, programcının çözümü geliştirirken problem ve çözüm kümelerini nesneler ve bu nesneler arasındaki ilişkiler olarak tanımlaması ilkesine dayanır. Yani gerçek yaşamın belirli bir kesitinin, nesneler ve nesneler arasındaki ilişiler olarak program birimlerine dönüştürülmesidir.

Bu başlıkta NYP'nin tarihçesi, en temel öğeleri olan nesne ile nesneler arasındaki ilişkiler ve NYP kavramları ele alınmıştır.

4.1. Nesneye Yönelik Programlamanın Tarihçesi

Daha öncesinde de çeşitli çalışmalar olmakla beraber, nesnelerin programlama varlıkları olarak ilk kez tanıtılması 1960 yılında Ole-Johan Dahl ve Kristen Nygaard tarafından Simula 67 programlama diliyle birlikte gündeme gelmiştir. Gemi simülasyonları tasarlamak için geliştirilen bu dille birlikte nesneler ve örnekleri (instance) kullanılmıştır. [24]

1970 de Xerox PARC tarafından tanıtılan Smalltalk ile nesneye yönelik programlama, "bilgisayar tabanlı hesaplamanın nesneler ve mesajlarla gerçekleşmesi" ifadesiyle gündeme gelmişti. Smalltalk tasarımcıları temel olarak Simula 67'den etkilenmişler ancak ondan farklı olarak sınıfların oluşturulmasını ve kullanımı dinamik hale getirmişlerdir. Ardından 1970'lerde Lisp, Modula-2 gibi diller gelmektedir. 1979 yılında C programlama dili Bell Laboratuvarları tarafından "Sınıflı C – Objective C" orijinal adıyla C++ programlama dilini geliştirmiştir. [24]

1980'li yıllardaysa birçok yeni NYP dili geliştirilmiş, var olan dillere NYP desteği eklenmiştir. NYP özellikleri eklenen dillerden bazıları ADA, Basic, Fortran ve Pascal'dir.

Diğer taraftan Pyhton ve Ruby gibi NYP için geliştirilen bazı diller de aynı zamanda prosedürel programlamaya destek vermek üzere tasarlanmışlardır.

1990'lara gelindiğindeyse, NYP desteği ötesinde programlama çerçevesi (framework) üzerinde çalışan Java, C# ve Visual Basic.NET gibi diller yazılım geliştiricilere sunulmuştur.

4.2. Nesneye Yönelik Programlamanın Temel Kavramları

NYP'nin en temel kavramları olan nesne, sınıf, kalıtım, çok biçimlilik, kapsülleme gibi kavramların iyi anlaşılması, NYP'nin iyi anlaşılması için önemlidir. Bu başlık altında bu kavramlara değinilmiş ve aralarındaki ilişkiler açıklanmıştır.

4.2.1. Nesne (Object)

"Nesneler belirli bir duruma, iyi tanımlanmış belirli tutumlara ve onları diğerlerinden ayıran bir kimliğe sahip olan varlıklardır." [25]

"Nesne, bir dizi aktiviteden oluşan bir "şey"dir. Bu bir dizi aktivite, nesnenin tutumlarını belirler. Bir nesne başka bir nesneye ne yapacağını mesaj yollayarak söyler." [26]

Nesneye yönelik programlama yaklaşımının temelini "nesne" kavramı oluşturmaktadır. Nesne, kendine has özellikleri olan ve özelliklere bağlı olarak belirli bir durumda bulunarak çeşitli tutumlar sergileyen somut ya da soyut varlıklardır. Bu tanım, "somut" kısmıyla gerçek yaşamdaki nesneler için de geçerlidir. Ancak gerçek yaşamda "nesne" olarak nitelendirmediğimiz işlem, sipariş, talep gibi kavramlar programlama ortamında nesne olarak tanımlanabilirler. Bu da tanımın "soyut" diye ifade ettiği bölümü oluşturur.

Aşağıda nesnelerin temel bileşenleri olan durum, tutum ve kimlik kavramlarına değinilmektedir.

Durum (state):
Her nesnenin bir "durumu" vardır. Durumla ifade edilmek istenen şey, nesnenin özelliklerinin belirli bir andaki değerlerinin tümüdür. Özellikler nesnenin karakteristiğini, kalitesini, vasıflarını belirtir. Örneğin bir masanın metal, ahşap ya da cam oluşu o masının özelliklerinden biridir ve masayı nitelendirir. Bir masanın yükseklik, en, boy, biçim, ağırlık, malzeme türü gibi birçok niteliği olabilir. Eğer bir masa nesnesi bu bahsettiğimiz niteliklere sahipse, bu niteliklerin her biri için bir de değere sahip olmalıdır. Örneğin;

Masa	A	B
Yükseklik	125 cm	100 cm
En	75 cm	75 cm
Boy	100 cm	100 cm
Biçim	Yuvarlak	Kare
Ağırlık	30 kg	30 kg
Malzeme Türü	Ahşap	Metal

Şekil 4-1 Nesneler ve nitelikleri

Görüldüğü gibi masa nesnesinin nitelikleri A ve B olarak adlandırılan iki ayrı örnek için farklı değerlere sahiptir. Bunlardan bir kısmının aynı olması farklı üyeler oldukları gerçeğini değiştirmez.

Kisi Sınıfı	A Nesnesi	
	1.1.2009	1.1.2010
TC Kimlik No	12345678901	12345678901
Ad	Mina	Mina
Yaş	8	9
Kilo	30 kg	40 kg
Boy	140 cm	150 cm

Şekil 4-2 Nesne ve farklı durumları (state)

Diğer taraftan bir nesnenin durumu, yani özelliklerinin değerleri zaman içinde değişebilir. Bu durum nesnenin aynı nesne olması gerçeğini değiştirmez.

Tutum:

Tutum, nesnelerin kendilerine gelen mesajlara verdikleri cevaplar ve etkilere gösterdikleri tepkilerden oluşur. Tutum o nesnenin "yapabileceklerinin/davranışlarının" görünür olanlarıdır. Tutumlar programlama dillerinde kendilerini metotlar ve olaylar (events) olarak gösterir.

Kimlik:

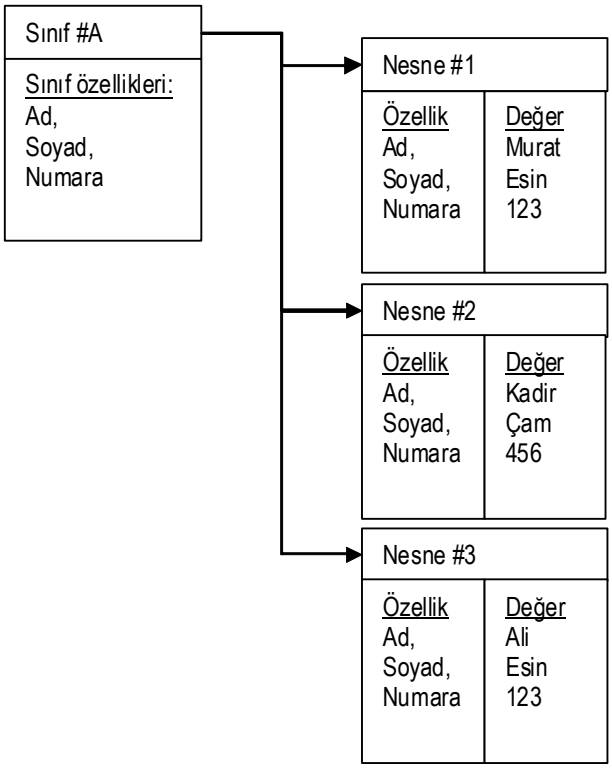
Kimlik (identity), nesneyi diğer örneklerinden ayıran özelliğidir. Bir nesnenin tüm örnekleri aynı şablondan oluştuklarından aynı özelliklere sahiptirler. Ancak bu özelliklerden bir tanesi, alacağı değerler bakımından örneği diğerlerinden ayırabilir olmalıdır. Böylece o sınıftan oluşturulmuş nesnelerin birbirlerinden ayırıcı birer kimlikleri oluşturulabilir. Buna benzer bir durum ilişkisel veritabanlarındaki kayıtlar için geçerlidir. Veritabanlarında bu işi gerçekleştirmek için birincil anahtar alanları kullanılır. Böylece tablo içinde birbirinin aynı değere sahip iki satır oluşması engellenir. Nesne oluştururken de eğer kimlik oluşturacak bir tasarım/modelleme yapılmamışsa, aynı sınıftan oluşturulan ve tüm özellikleri aynı değerde olan iki ya da daha fazla nesne tanımlanması mümkün olabilecektir. Bu durumda bir modelleme/tasarım hatası söz konusu olmaktadır.

4.2.2. Sınıf

Gerçek yaşam perspektifinden bakıldığında sınıf, aynı türden nesnelerin oluşturduğu bir kümedir. Etrafımızda birçok araba görürüz ve bu arabaları "araba" olarak sınıflandırarak konuşur, tanımlarız. Benzer şekilde ev, yol, semt gibi birçok türde nesneyi bir arada temsil ederek nitelendiririz.

Programlama perspektifinden bakıldığındaysa sınıf, bir nesnenin özelliklerini ve davranışlarını tanımlayan şablondur; nesnenin kodlanmış halidir ve geliştirme zamanıyla ilgilidir. Kullanılan programlama diline bağlı olarak farklılık gösterse de, nesnenin durumunu taşıyacak olan özelliklerin neler olacağı, hangi veri tiplerinde oldukları; nesnenin tutumunu belirleyecek olan olayların ve metotların isimleri, alacakları parametreler ve çağırıldıklarında ne yapacakları sınıf içinde kodlanarak tanımlanır.

Gerçek yaşamda her nesne bir sınıfın üyesi olarak karşımıza çıkmaktadır. Programlama açısından bakıldığındaysa nesneler sınıf şablonu kullanarak bellekte yapılandırılırlar. Bu işleme sınıfın ya da nesnenin bir örneğinin türetilmesi (instanciating) adı verilir.



Şekil 4-3 Sınıftan Nesne Türetme

Yukarıdaki şekil 5.1 de Ad, Soyad ve Numara niteliklerine sahip bir A sınıfından oluşturulmuş 1, 2 ve 3 numaralı üç farklı nesne görmektesiniz.

4.2.3. Sınf ve Nesne Kavramlarının İlişkisi

NYP'de bir nesneden bahsedildiğinde, gerçek yaşamdaki bir nesnenin, belirli bir uygulama (çözüm alanı) içindeki soyutlanmış hali kastedilir. Yani bir adres defteri uygulamasındaki "Kisi" nesnesinden bahsediyorsanız, bu nesnenin soyutlanmış halinde isim, soyad, telefon numarası, adres gibi özellikler olacaktır. Ancak kan grubu, sosyal sigorta numarası, mezuniyet derecesi gibi gerçek yaşamdaki bir "Kişi" nesnesinin sahip olabileceği yüzlerce özellik bu soyutlamanın dışında bırakılacaktır.

Uygulama içinde nesnenin bu özellikleri, özelliklere bağlı olarak gerçekleştireceği tutumlar yani işlemler, tepkiler kodlanarak bilgisayarın bu nesneyi nasıl oluşturacağı tanımlanır. İşte bu nesnenin kodlanmış tanımına "sınıf" adı verilir. NYP dillerinden biriyle bir uygulama gelişti-

riyorsanız ve uygulama içinde bir nesneye ihtiyacınız varsa, bu nesne için bir sınıf kodlamanız gerekir.

Bir NYP uygulamasının çalışması esnasında birçok sınıftan türemiş nesneler bir arada çalışıyor olabilir. Hatta aynı anda bu sınıflardan bazılarına ait birden fazla aynı türden nesne olabilir. Böyle bir durumda farklı sınıflardan türetilmiş olan nesneleri birbirlerinden ayıran şeyler, bu nesnelerin özellik ve davranış setleri olacaktır. Aynı türden nesneleri birbirinden ayıran şeyse, aynı özellikte olmalarına rağmen durumlarının (state) yani özelliklerinin taşıdığı değerlerin farklı oluşudur.

4.2.4. Metot

Bir nesnenin yapabileceklerini, o nesnenin metotları belirler. Dışarıdan erişilebilir bir arayüze sahip olan ve fonksiyon olarak tanımlanan isimlendirilmiş kod blokları, nesnenin metotlarıdır. Metotlar parametreli ya da parametresiz olarak tasarlanabilirler. Parametresiz bir metot, sadece ismi çağırılarak kullanılabiliyorken, parametreli bir metodu çağırmak için, metodun beklediği parametreleri uygun şekilde yollamak gerekir. Bazı metotlar çağırıldıklarında sadece tanımlanmış belirli bir işlevi yerine getirirler. Bazı metotlarsa kendilerini çağıran nesneye geri yönde bir mesaj yollarlar.

4.2.5. Mesaj İletimi

Bir nesnenin başka bir nesneye bilgi göndermesi, o nesneden bilgi istemesi ya da bir iş yapmasını söylemesi mesaj iletimi olarak nitelendirilmektedir. Bu, programlama açısından bir nesnenin diğer metodu çağırmasıdır. Bir nesne diğer bir nesnenin metodunu çağırırken, ona en azından metodun adını mesaj olarak iletir. Eğer çağırılan metot, parametreli bir metotsa, bu sefer mesajı iletecek nesne, mesajı alacak olan nesneye, metodun adıyla birlikte belirlenmiş veri tiplerinde parametre değerlerini de içeren daha kapsamlı bir mesaj yollar. Eğer metot geri dönüşlü bir metotsa, bu sefer çağırılan nesne, çağıran nesneye bir mesaj yollayacaktır. Mesaj iletimi, nesneler arasındaki iletişim şeklidir.

4.2.6. Kalıtım

Kalıtım en basit tanımıyla, bir sınıfın belirli özellik ve tutumlarını bir üst sınıftan alarak, kendisi için farklı olan özellik ve tutumları ayrıca uyguluyor olmasıdır. Bu tanımdaki bir üst sınıf ilişkisi, genel sınıflandırma mantığı içinde bir hiyerarşiyi ifade eder. Bunu bir örnekle anlatacak olursak, bir kedi, üst sınıf olarak "kedigiller"e aittir. Bunun da üzerinde etoburlara, memelilere, omurgalılara ve en üstte hayvanlar sınıfına dâhildir. Kedi, en üstteki "hayvan" sınıfından başlayarak her bir alt sınıfta bir takım özellikleri özelleşerek, en sonunda kendi sınıfına ait özelliklere sahip olur.

Programlama açısından kalıtım, sınıfın özelliklerinin üst sınıflara ait genel olanları yeniden kodlamaya gerek duymaksızın, üst sınıflardan devralması anlamına gelmektedir. Böylece genel özellikler ve tutumlar üst sınıflardan (parent) gelirken, yazılımcı sadece sınıfa ait olan durum ve tutumları programlar. Bu durumda belirli durum ve tutumları miras alınan sınıfa ebeveyn (parent) ya da süper (super) sınıf denir. Üst sınıfın bir alt sınıfı yani mirasçısı olarak tanımlanan yeni sınıfa da alt sınıf (subclass) adı verilir.

Örneğin Kaynak adında bir sınıfı ele alalım. Bu sınıfın amacı bir kütüphanedeki öğelerin tümünü temsil etmek olsun. Kütüphanedeki her bir öğe için, onun yerini ve kategorisini belirleyen bir kodu olacaktır. Bu durumda "Kod" özelliği Kaynak nesnesinin kimlik özelliği olarak karşımıza çıkar. Ayrıca bu kaynağın kütüphanedeki yerini belirleyen bir "Yer" özelliği de olabilir. Sonuçta elimizde kod ve yer özelliği olan bir sınıf bulunmaktadır. Şimdi kütüphanedeki kitaplar için kaynak sınıfından türetilmiş bir "Kitap" sınıfı oluşturulduğunu düşünün. Bu Kitap sınıfının doğal olarak ebeveyninden miras aldığı bir Yer ve Kod özelliği olacaktır. Buna ek olarak kitap türündeki kaynakla riçin yazar, yayınevi, sayfa sayısı, basım sayısı vb. gibi bilgiler tutmak istenebilir. İşte bu durumda bu yeni özellikler, yeni sınıfa ait, özelleştirilmiş üyelerdir.

Java ve C# gibi bazı programlama dilleri tek bir sınıftan miras almayı desteklerken C++ birden fazla sınıftan miras almayı destekler.

4.2.7. Soyutlama

Nesnenin uygulama için gerekli olan özelliklerinin kodlanarak, diğer özelliklerinin kodlanmaması uygulamaya dahil edilmemesi anlamına gelir. Örneğin bir otopark yazılımda aracın plaka bilgisi, giriş/çıkış saatleri önemlidir. Eğer marka/modele göre ücretlendirme yapmıyorsanız, bu bilgilerin sınıfa/nesneye dâhil edilmesine gerek yoktur. Ayrıca aracın şase numarası, motor hacmi, motor numarası gibi büyüklükler de soyutlamanın dışında bırakılmalıdır.

4.2.8. Çok Biçimlilik

Çok biçimlilik, nesnelerin kendi türlerinin dışında farklı türler olarak da temsil edilmelerini sağlar. Ancak bu alt-sınıf/üst-sınıf ilişkisiyle sınırlıdır. Yani belirli bir üst sınıftan türetilmiş olan bir sınıf, her ne kadar kendi özel tutum ve durumlarına sahip olsa da, üst sınıfından bir nesneymiş gibi de davranabilirler.

Bir önceki başlıkta anlatılan kitap ve kaynak sınıfları üzerinden devam edecek olursak, bir kitap nesnesi aynı zamanda bir kaynak nesnesi olarak da kullanılabilir. Bunu sağlayan, kitap sınıfının kaynak sınıfının bir alt sınıfı olmasıdır. Böylece kitap sınıfı hem kitap hem de kaynak sınıfı olarak ifade edilebildiğinden çok biçimlilik göstermiş olur.

4.2.9. Kapsülleme (encapsulation)

Bilgi saklama ya da koruma olarak da bilinen kapsülleme, nesnenin yaptığı işler ve durumuyla ilgili bilgilerin ve işlevsel karmaşıklığının gizlenmesi anlamına gelir. Tanımından da anlaşılacağı üzere iki temel amacı vardır: bilgi saklama ve karmaşıklığı gizleme.

Kapsülleme sadece bilginin yani nesnenin özelliklerinin değerlerinin saklanması değil, aynı zamanda hangi metotlarının erişilebilir olup olmayacağının da belirlenebilmesi anlamına gelmektedir.

Kapsüllemenin sonucu olarak, her sınıf için dışarıya açılacak metotlarının bir şekilde bildirilmesi gerekir. Sınıfın arabirimi (interface) olarak isimlendirilebileck olan bu tanımlamayla, diğer nesnelerin bu nesne üzerinden hangi metotlara ulaşabileceği belirlenmiş olur. Birçok

programlama dilinde, arabirimlerin hangi seviyelerden (sadece o sınıfın dahil olduğu proje, ya da herkes tarafından gibi) erişilebilir olduğu belirlenebilmektedir.

4.2.10. Nesneler Arasındaki İlişkiler

Nesneler arasındaki ilişkiler farklı şekillerde karşımıza çıkmaktadır. Bunlardan en çok karışlanılanları bileşim (composition), kümeleme (aggregation), ortaklık (association), alt sınıf / üst sınıf (super-class / sub-class) ve kullanım (using) olarak sıralanabilir. Nesneler arasındaki ilişkileri tanımlarken ve karşılaştırırken iki temel ölçü kullanılmıştır; sahiplik ve yaşam döngüsü.

Bileşim İlişkisi:
Bileşim, bir nesnenin bir ya da daha fazla nesnenin bileşimiyle oluşması anlamına gelmektedir. Örneğin bir araba nesnesini ele alırsak, araba nesnesi motor, yönlendirme mekanizması (direksiyon ve bağlı mekanizma), şasi, kasa, vb. nesnelerden oluşur. Bu nesnelerden herhangi biri olmaksızın, araba nesnesinin varlığından bahsedilemez. Bileşim ilişki tipinde, esas nesne tüm diğer bileşenleri oluşturan nesnelerin sahibidir. Bileşeni oluşturan nesneler, esas nesne oluştuğunda var olurlar ve esas nesne yok olduğunda onların da yaşam döngüsü sonlanmış olur. Örneğin bir araba nesnesini kasa, şasi ve motor nesneleriyle oluşturup, araba nesnesinin yaşam döngüsünü sonlandırdığınızda, artık kasa, şasi ve motor nesneleri de yaşam döngüsünü sonlandırmış olacaktır. Bileşim ilişkisinin bir diğer belirleyici unsuru da, nesnenin bileşenlerinin başka bir nesne tarafından kullanılamamasıdır. Yani bir şasi nesnesi sadece tek bir araba nesnesiyle ilişkili olabilir.

Kümeleme İlişkisi:
Kümelemede iki nesnenin arasındaki ilişki, bir nesnenin diğer nesneyi içermesi şeklindedir. Örneğin bir araba yolcu taşır. Ancak yolcu taşımayabilir de. Yolcu nesnesi araba nesnesi için mutlaka olması gereken bir yapı taşı değildir. Bileşimle kümelemenin farklılaştığı nokta da burasıdır. Yaşam döngüsü açısından her nesne kendi yaşam döngüsüne sahiptir. Yani araba ve yolcu nesneleri ayrı ayrı oluşturulabilir. Daha sonra yolcu nesnesi belirli bir araba nesnesinin üyesi olarak atanabilir. Bu durumda araba nesnesinin yaşam döngüsü sona erse bile, yolcu nesnesinin yaşam döngüsü devam edebilecektir. Sahiplik açısından bakıldığındaysa araba nesnesi yolcu nesnesinin sahibi olacaktır. Ancak yolcu nesnesinin tek bir sahibi olabilir.

Ortaklık İlişkisi:

Ortaklık ilişki türünde, tüm nesneler kendi yaşam döngülerine sahiptirler ve bir nesnenin diğerine sahipliği söz konusu değildir. Araba örneğinden devam edecek olursak, araba nesnesi ile otopark nesnesi arasındaki ilişkide, bir otopark birçok arabayla ilişkili olabilir. Bu çalışma ortaklığında nesneler ayrı ayrı oluşturulmuştur ve otopark arabanın ya da araba otoparkın sahibi değildir.

Alt Sınıf / Üst Sınıf İlişkisi:

Alt-sınıf/üst-sınıf ilişkisi kalıtımla ilgilidir. Bir nesne, başka bir nesnenin alt sınıfı olabilir. Örneğin otomobil, minibüs, otobüs ve kamyon, taşıt sınıfının alt sınıfı olarak karşımıza çıkar. Üst sınıf ile alt sınıf arasında kalıtım nedeniyle özellik ve davranışların paylaşılması söz konusudur.

Kullanım İlişkisi:

Kullanım ilişkisi ise daha çok kodlama seviyesiyle ilişkilidir. Yukarıdakilerin hepsinin dışında bir yöntem olarak, nesnenin metotlarından biri, parametre olarak temel veri tipleri yerine bir başka nesneyi kullanıyor olabilir. Böylece metoda çok sayıda parametre göndermek yerine tek bir öğe gönderilmiş olur. Ayrıca ilgili metot içinde parametre olarak alınan nesnenin işlevlerinden yararlanma olanağı de elde edilmiş olmaktadır. Bu durumda parametre olarak kullanılan nesne, içinde bulunduğu sınıfa ait olur. Çalışma zamanıysa metodun çağırıldığı andan, görevini tamamlandığı süre kadar olacaktır.

5. ETMENE YÖNELİK PROGRAMLAMA

İş dünyasının dinamik ve hızlı gelişen yapısı, gittikçe karmaşıklaşan süreçleri doğurmuş ve bu süreçleri otomasyonla yönetmek için geliştirilecek yazılımların da karmaşıklığı ve büyüklüğü gittikçe artmaya başlamıştır. Mantıksal ve prosedürel programlamaya göre daha verimli görünen NYP, büyük ve karmaşık program geliştirmenin gerektiği durumlarda istenileni verememeye başlamıştır.

Temel olarak nesnelere ve nesneler arasındaki ilişkilere dayalı olarak geliştirilen NYP uygulamaları, kendi başına çalışması gereken, belirli amaçları olan ve etrafındaki değişimleri algılayarak, uygun planlar üzerinden iş yapacak birimler oluşturmada yetersiz kalmaktadır. NYP ile geliştirilen yazılımlar doğaları gereği pasiftir ve dışarıdan bir müdahaleyle çalışırlar.

Yukarıdaki ihtiyaçla ilgili güzel bir örnek, aidatını ödemeyen üyelere ilk olarak eposta gönderen, aradan beş iş günü geçtikten sonra halen aidatını ödemeyen üyelere kısa mesaj yollayan ve aidatın ödeme süresini 15 gün geçen üyeleri, sistem yöneticisine eposta yoluyla bildiren bir aidat yönetim sistemi bileşeni verilebilir. Böyle bir işlev, üyelerin ödeme bilgilerinin takip edilmesini, belirli zaman aralıklarının kontrol edilmesini, süreç içindeki değişimlere göre farklı planların uygulanmasını gerektirir. Hata böyle bir sistemden belirli bir süre sonra, aidatını geciktirmeyi alışkanlık edinmiş üyelere eposta seçeneğini atlayarak direkt olarak kısa mesaj göndermesi beklenebilir.

NYP tüm bu işleri gerçekleştirecek ve amaçlara ulaşılmasını sağlayacak işlev ve özellikleri geliştirmek için kullanılabilir. Ancak NYP ile geliştirilen program sürekli olarak bir kullanıcının belirli işlevleri çalıştırmasını gerektirecektir.

Bu tip ve benzeri iş dünyası/kullanıcı istekleri, NYP ile birlikte yeni bir program geliştirme paradigmasının daha kullanılmasını zorunlu hale getirmiş ve

“Etmene Yönelik Yazılım Mühendisliği (Agent-Oriented Software Engineering) kavramı Shoham tarafından 1993 yılında ortaya atılmıştır.” [19]

Etmene Yönelik Programlama (Agent-Oriented Programming), Nesneye Yönelik Programlamanın iyileştirilmiş bir uzantısı olarak karşımıza çıkmaktadır. NYP (Nesneye Yönelik Programlama) daha çok programlama dillerine ve uygulama seviyesine yakın bir kavramken, EYP daha üst seviye bir bileşen olgusunu temsil eder.

EYP ile anlamca benzeşen diğer terimler aşağıdaki gibidir: [20]

- Agent-Based Computing: Etmen Temelli Hesaplama
- Software Engineering with Agents: Etmenlerle Yazılım Mühendisliği
- Agent-Oriented Development (AOD): Etmene Yönelik Yazılım Geliştirme
- Agent-Based Software Engineering: Etmen Temelli Yazılım Mühendisliği
- Multi-agent Systems Engineering (MaSE): Çok-etmenli Sistemler Mühendisliği
- Agent-Oriented Software Engineering (AOSE): Etmene Yönelik Yazılım Mühendisliği

Etmene Yönelik Yazılım Mühendisliğinin amacı, uygun maliyetle kaliteli etmene yönelik yazılım geliştirmek için gerekli metodoloji ve araçları oluşturmaktır.

Gerçek yaşamdaki karmaşık problemleri çözmek, o problemleri gerçek yaşamdaki en yakın şekliye soyutlamakla mümkün olacaktır. Etmene yönelik programlama gerçek yaşamdaki bir öğeyi ayrı bir etmen olarak ele alır ve bu öğelerin birbirleriyle olan ilişkilerini de etmenler arasındaki iletişim ve birlikte çalışma olarak tanımlar. Böylece daha önceki yazılım geliştirme yaklaşımlarının çözemediği karmaşıklıktaki problemleri çözebilir.

5.1. Etmene Yönelik Yazılım Geliştirme Yaşam Döngüsü

Jennigs ve Wooldridge etmene yönelik yazılım geliştirmenin yaşam döngüsünü üç temel adımda tarif ederler. [21] Bunlar şöyledir:

1. Belirtim (specification),
2. Uygulama (implementation),
3. Doğrulama (verification)

Belirtim aşamasında geliştirilecek olan etmenle ilgili gereksinimlerin neler olduğu, nasıl bir altyapıya ihtiyaç duyulduğu, etmenin ne tür özellikleri olması gerektiği gibi sorulara cevap aranır. Etmenler için bu soruların cevapları, geliştirilecek olan etmenin inanç, hedef, eylem ve etkileşim yetenekleri perspektifinden verilmelidir.

Belirtim oluşturulduktan sonra sıra uygulama aşamasındadır. Bu noktada belirtimde tespit edilenlerin gerçekleştirilmesi için iki farklı yol izlenebilir. Bunlardan birincisi belirtimi direkt olarak belirli bir dille kodlayarak geliştirmektir. Bu noktada etmen tabanlı yazılım geliştirme dil ve altyapıları kullanılabileceği gibi (AGENT0), geliştirilecek etmene uygun olması durumunda başka herhangi bir dil ya da teknoloji de tercih edilebilir. Diğer bir uygulama yöntemi de, belirtim üzerinden direkt olarak etmenleri oluşturacak araçların kullanılmasıdır. Örneğin "Concurrent MetateM" gibi araçlarla belirtim özel bir yöntemle sentezlenir ve otomatik olarak etmenlere dönüştürülebilir. Böylece çok daha hızlı bir şekilde etmen oluşturma imkânı doğar. Ancak etmenlerin yetenekleri ve öğrenebilirlikleri sınırlı olacaktır.

Yaşam döngüsündeki son aşama doğrulama aşamasıdır. Bu aşamada geliştirilmiş olan sistemin, belirtim aşamasındaki gereksinimleri karşılayıp karşılamadığı kontrol edilir.

5.2. Etmen Geliştirme Standartları

Etmenler üzerine geliştirilen ve yaygın olarak kabul görülen temel standartlar için IEEE organizasyonu tarafından The Foundation for Intelligent Physical Agents (Akıllı Fiziksel Etmenler için Temeller) komisyonunu kurmuştur. Bu komisyonun amacı, akıllı etmen geliştirmeyle ilgili temel ilke, yöntem ve yaklaşımları belirlemek; akıllı etmenlerin diğer teknolojilerle entegrasyonunun nasıl sağlanacağı üzerine çalışmalar yapmaktır.[22]

Aşağıda FIPA tarafından geliştirilen akıllı etmen geliştirmeye yönelik standartların listesi yer almaktadır. [22]

- FIPA Abstract Architecture Specification (Soyut Mimari Belirtimi)
- FIPA SL Content Language Specification (İçerik Dili Belirtimi)
- FIPA Nomadic Application Support Specification (Göçebe Uygulama Destek Belirtimi)

- FIPA Agent Management Specification (Etmen Yönetim Belirtimi)
- FIPA Request Interaction Protocol Specification (İstek Etkileşimi Protokol Belirtimi)
- FIPA Query Interaction Protocol Specification (Sorgu Etkileşimi Protokol Belirtimi)
- FIPA Request When Interaction Protocol Specification (Etkileşim Olduğunda İstekte Bulunma Belirtimi)
- FIPA Contract Net Interaction Protocol Specification (İnternet Etkileşim Kontrat Protokol Belirtimi)
- FIPA Iterated Contract Net Interaction Protocol Specification (Yinelemeli İnternet Etkileşim Kontrat Protokol Belirtimi)
- FIPA Brokering Interaction Protocol Specification (Aracı Etkileşimi Protokol Belirtimi)
- FIPA Recruiting Interaction Protocol Specification (İşe Alma Etkişelimi Protokol Belirtimi)
- FIPA Subscribe Interaction Protocol Specification (Abone Olma Etkileşimi Protokol Belirtimi)
- FIPA Propose Interaction Protocol Specification (Önerme Etkileşimi Protokol Belirtimi)
- FIPA Communicative Act Library Specification (İletişime Açık Eylemler Protokol Belirtimi)
- FIPA ACL Message Structure Specification (Mesaj Yapı Belirtimi)
- FIPA Agent Message Transport Service Specification (Etmen Mesaj Aktarım Servisi Belirtimi)
- FIPA ACL Message Representation in Bit-Efficient Specification (Bit-Verimli Mesaj Temsil Belirtimi)
- FIPA ACL Message Representation in String Specification (Mesaj Metin Temsil Belirtimi)
- FIPA ACL Message Representation in XML Specification (Mesaj XML Temsil Belirtimi)
- FIPA Agent Message Transport Protocol for IIOP Specification (IIOP İçin Etmen Mesaj Aktarımı Belirtimi)
- FIPA Agent Message Transport Protocol for HTTP Specification (HTTP İçin Etmen Mesaj Aktarım Belirtimi)

- FIPA Agent Message Transport Envelope Representation in XML Specification (XML Etmen Mesaj Aktarım Zarf Temsili Belirtimi)
- FIPA Agent Message Transport Envelope Representation in Bit Efficient Specification (Bit-Verimli Etmen Mesaj Aktarımı Zarf Temsili Belirtimi)
- FIPA Device Ontology Specification (Cihaz Oluşturma Belirtimi)
- FIPA Quality of Service Specification (Servis Kalite Belirtimi)

Bunun dışında akıllı etmenlerle ilgilenenler için çeşitli bağımsız inisiyatifler ve konsorsiyumlar kurulmuştur. Bu organizasyonların amacı çeşitli çalıştaylar, konferanslar ve etkinlikler düzenleyerek akıllı etmen teknolojilerini geliştirmektir.

Bu organizasyonlardan bazıları şöyledir:

- OMG (Object Management Group)'un kurduğu OMG Agent Platform Special Interest Group, http://www.objs.com/agent/
- The Web Intelligence Consortium, http://wi-consortium.org/
- Agent UML, http://www.auml.org/

OMG etmen platformu, etmen teknolojilerin desteklemek ve etmene yönelik programlamaya yönelik analiz, tasarım ve geliştirme süreçleri için çeşitli mimariler, araçlar ve yöntemler araştırmak ve geliştirmek için kurulmuştur. OMG bu amacını gerçekleştirmek için çeşitli araştırmalar ve yayınlar yapmakta, etkinlikler düzenlemektedir.

Web zekâsı konsorsiyumu ise, yapay zekâ ve ileri bilgi teknolojilerini kullanarak yeni nesil web tabanlı ürünler, sistemler, servis ve aktiviteler geliştirmek üzere kurulmuş, kar amacı gütmeyen bir organizasyondur. Misyonunu gerçekleştirmek için bilimsel araştırmalar, endüstriyel çalışmalar yapmakta, ilgili konferanslara katılmakta ve dergi, gazete ve makale yayınlamaktadır.

Agent UML, UML (Unified Modeling Language) 'i temel alan, etmene yönelik programlama için genel bir modelleme yaklaşımı geliştirmek üzere çalışan bir gruptur. Çoklu etmen mimarisine dayalı sistemlere özel durumları belirleyerek, bu durumları destekleyebilen bir modelleme yapısı geliştirme üzerine, platform bağımsız bir çözüm geliştirmeye çalışmaktadırlar.

5.3. Etmen Geliştirme Dilleri, Metodolojileri, Ortamları ve Altyapıları

Yapılan çalışmalar, çoklu-etmen programlama üzerine deklaratif, imperatif ve bu ikisinin hibritleri de dahil olmak üzere pek çok etmen yönelimli programlama dili olduğunu ortaya çıkartmıştır. Bu dillerin bir kısmı tamamen baştan geliştirilmiş olduğu gibi, bazıları da daha önceden var olan yapısal ve nesneye yönelik programlama dillerinin üzerine inşa edilmiştir.

Etmene yönelik programlama için geliştirilen diller, çoğunlukla bir platform ya da altyapı ile birlikte kullanılırlar. Bununla birlikte etmenlere yönelik platformlar geliştirilirken daha çok etmenler arasındaki iletişim ve koordinasyonun sağlanması amaçlandığından, belirli bir programlama diline bağlı olmaması esastır. [23]

Diğer taraftan etmene yönelik uygulama geliştirmek için yazılımcılar için gerekli olan en önemli araçlardan birisi de yazılım geliştirme ortamıdır. Dili ve platformu en iyi şekilde kullanarak etmen geliştirmek, derlemek ve test etmek için gerekli olan ortamlar, çoğunlukla programlama dili ile birlikte gelirler.

Bunların ötesinde, karmaşık problemlerin çözümünde kullanılacak olan etmenlerin, platform, altyapı, programlama dili ve geliştirme ortamı kullanılarak nasıl gerçekleştirileceği, gereksinimlerin nasıl belirleneceği, tasarımın nasıl yapılacağı ve uygulanacağı, testinin nasıl gerçekleştirileceği ile ilgili çalışmalar da etmene yönelik programlama metodolojilerinin gelişmesini sağlamıştır.

5.3.1. Bildirimsel (Declarative) Diller

Bildirimsel diller, programcının programın işlemi nasıl yapacağından çok, ne yapacağını bildirmesi esasına dayanan dillerdir. Etmen yönelimli programlama için geliştirilen bildirimsel diller aşağıda incelenmiştir.

CLAIM (Computational Language for Autonomous, Intelligent and Mobile Agents)
Otonom, akıllı ve mobil etmenler için hesaplama dili olarak ifade edilen bu dil, yüksek seviyeli bildirimsel bir etmene yönelik programlama dilidir. Dil HIMALAYA (Hierarchical Intelligent Mobile Agents for building Large-scale and Adaptive sYstems based on Ambients)

adı verilen bir altyapıyla birlikte gelmektedir. CLAIM SyMPA adı verilen dağıtık sistem platformu ile de birlikte çalışabildiğinden, dağıtık mobil etmenler geliştirmek için de kullanılabilir. [23]

FLUX

FLUX yukarı seviye bir bildirimsel etmen programlama sistemidir. "Fluent Calculus" yaklaşımının bir uygulamasıdır. [23]

MINERVA

Mantıksal programlama temeline dayanan genel bir etmen yönelimli programlama sistemidir. Etmenleri tanımlamak ve hareketlerini belirlemek için MDLP ve KABUL kullanılır. [23]

MDLP (Multi-Dimensional Dynamic Logic Programming)

Çok boyutlu dinamik mantık programlama olarak ifade edilen MDLP, MINERVA üzerinde uygulama geliştirmek içini kullanılan bir dildir. ASP (Answer Set Programming) dilinin bir uzantısıdır. [23]

KABUL (Knowledge And Behavior Update Language)

Bilgi ve davranış güncelleme dili olarak tanımlanan KABUL, EVOLP dilinin evrimleşmiş halidir. [23]

DALI

DALI bir aktif mantık programlama dilidir. Direkt olarak derlenebilecek belirtimler oluşturmak için kullanılır. [23]

ReSpecT

Mantık tabanlı bir bildirimsel dildir. Reaksiyonların ve kuralların tanımlanabilmesi için iyi tanımlanmış bir biçimsel yapıya sahiptir. TuCSoN altyapısı üzerinde çalışır.

5.3.2. İmperatif (Imperative) Diller

İmperatif diller, etmen yönelimli programlar geliştirilirken, deklaratif dillere oranla daha az kullanılmaktadır. İmperatif dillerde, deklaratif dillerden farklı olarak programcının etmen,

çevresiyle ilişkisi, iletişim ve koordinasyon, olaylar ve tepkiler, karar yapıları, inançlar ve bilgilerin saklanması gibi tüm detayları programlaması gerekmektedir.

İmperatif olarak programlama geliştirileceğinde, programcının kullanacağı dilin özellikle etmene yönelik programlamaya özgün olmasına gerek yoktur. Bu noktada herhangi bir yapısal ya da nesneye yönelik dille de etmen programlanabilmektedir.

Diğer taraftan bazı firmalar ve kurumlar tarafından etmene yönelik imperatif programlama dilleri geliştirilmektedir. Bu dillerden bir tanesi Agent Oriented Software firması tarafından geliştirilen The JACK Agent Language (JAL)'dır. JAL, Java üzerine geliştirilmiş bir dildir. [23]

5.3.3. Hibrid Yaklaşımlar

Bilinen birçok etmene yönelik programlama dili hem bildirimsel hem de imperatif yapıyı birleştiren hibrid çözümlerdir. Hibrid yaklaşımla geliştirilmiş olan dillerden bazıları şöyledir:

3APL:
(An Abstract Agent Programming Language "triple-a-p-l"), inanç, hedef, plan ve mental tavırları olan bilişsel etmenler geliştirmek için tasarlanmış bir dildir. En önemli özelliği mental tavırlar için kolaylıkla kullanılabilecek hazır yapılara sahip olmasıdır. Mental, harici ve iletişim aksiyonları için hazır bileşenleri vardır ve böylece tekrar kullanılabilirliği ve modülerliği sağlayarak etmen geliştirmeyi hızlandırır. 3APL ayrıca yüksek seviyeli robotlarda da kullanılmaktadır. [23]

Jason:
Mantık temelli etmene yönelik programlama dili olan Jason, AgentSpeak(L)'in gelişmiş bir versiyonudur. İnanç, amaç ve niyet mimarisine ve mantığına uygun olarak tasarlanmıştır. [23]

Go:
Go, çok farklı etmene yönelik programlama yaklaşımlarını destekleyen, bildirimsel birçok modülü ve hazır fonksiyonu olan, aynı zamanda ilişki tanımları ve karşılaştırmalar için

imperatif kodlamayı da destekleyen zengin bir dildir. Sembolik programlama dili olan April üzerine inşa edilmiştir.

AF-APL:
Agent Factory Agent Programming Language (AFAPL), Agent Factory yaklaşımının üzerin oturtulmuş bir dildir. Y. Shoham tarafından başlatılan çalışma daha sonraları BDI yaklaşımıyla zenginleştirilmiştir.

5.3.4. Geliştirme Ortamları

Entegre yazılım geliştirme ortamları, yazılım geliştirme sürecinin programlama (geliştirme) aşaması için oluşturulurlar. Nesneye yönelik programlama dilleri için geliştirilmiş olan yazılım geliştirme ortamlarına bakıldığında beş temel işlevi yerine getirdikleri söylenebilir: Proje yönetimi

1. Kaynak dosyalarını oluşturmak ve yönetmek
2. Yeniden düzenleme (refactoring)
3. Derleme ve çalıştırma
4. Test ve hata ayıklama

Ancak iş etmene yönelik programlamaya geldiğinde, kullanılacak olan altyapı ve programlama diline göre yapılacak işler değiştiğinden, oluşturulacak geliştirme ortamlarının tamamen bu dil ve altyapılara uygun olması gerekmektedir. Bu nedenle de dil ve altyapı/platform geliştiricilerinin kendi geliştirme ortamlarını da oluşturmaları beklenmektedir.

Aşağıda belirli dil ve platformlar için oluşturulmuş geliştirme ortamlarından bazıları incelenmiştir.

3APL IDE
Bu geliştirme ortamı çoklu etmenler için, etmenleri yönetmeye yarayan bir etmen yönetim sistemi, etmenlerin mesajlaşması için bir mesaj taşıma sistemi ve etmenlerin içinde bulundukları ortamda çeşitli aksiyonlar oluşturabilmeleri için bir "plug-in" arayüzüne sahiptir. Böylece 3APL etmenleri için bir işletim ve geliştirme ortamı sağlar.

Jason IDE

Çok-etmenli sistemler geliştirmek için bir grafik geliştirme ortamıdır. Bu ortam üzerinden etmenler ve sistem için yapılandırma dosyaları oluşturulabilir. Geliştirme ortamı üzerinden ayrıca etmenler kontrol edilebilir ve çalışma modları değiştirilebilir. Geliştirme ortamının sunduğu bir diğer avantaj da, etmenlerin mevcut durum bilgilerinin okunabilmesidir.

The JACK Development Environmet

Agent Oriented Software Ltd tarafından JACK etmenleri için oluşturulmuş bir geliştirme ortamıdır. Farklı bileşenler için proje dosyalarını bir ağaç yapısı şeklinde yönetmeyi destekler. Ek olarak bir de plan editörü vardır. Akış diyagramı yapısına benzer bir şekilde etmen planlarının geliştirilmesini destekler.

Visual Soar

Soar etmen mimarisi için geliştirilmiştir. Temel proje yönetimi ve Soar etmenlerinin programlanması için gerekli olan söz dizimi denetimi ve temel kontrol işlevlerini sağlar. Ek olarak Soar çalışma zamanına bağlanarak etmenlerin geliştirme ortamı üzerinden çalıştırılmaları da desteklenir.

AgentBuilder

Shoham'ın Agent0 dili üzerine geliştirilmiş olan Reticular Agent Language için oluşturulan bir geliştirme ortamıdır. Geliştirme ortamı çoğunlukla sihirbazlar ve grafik ekranlarla geliştiricinin etmenleri oluşturmasını ve düzenlemesini sağlar. Basit bir proje yönetimi işlevi ve derleyici aracı vardır.

The Agent Factory Development Environment

Temel proje yönetimi, düzenleme ve farklı etmen bileşenlerini birleştirme için destek verir. AF-APL'ye destek verir. Ayrıca etmene yönelik test ve hata ayıklamaya destek veren VIPER adında bir araç içerir.

5.3.5. Platformlar ve Altyapılar

EYP dillerinin birçoğu, etmene yönelik programlamanın mantığını destekleyen bir altyapı ya da platform üzerinde çalışmaktadır. Etmene yönelik programlamada daha çok, çoklu etmen

geliştirmede karşımıza çıkan altyapı ve platformların amacı etmenler arasındaki iletişimi ve koordinasyonu sağlamaktır.

Bu bölümde daha önceki iki bölümde bahsedilen dil ve ortamlarla birlikte çalışan altyapılardan bahsedilecektir.

TuCSoN

TuCSoN (Tuple Centre Spread over the Network) altyapısı çoklu-etmen programlamayı destekleyen genel amaçlı bir altyapı sağlar ve programlanabilir iletişim ve koordinasyon servislerine sahiptir. Merkezinde ReSPecT diliyle dinamik olarak programlanabilen bir koordinasyon çalışma zamanı bulunmaktadır.

JADE (Java Agent DEvelopment Framework)

Dağıtık çoklu-etmen uygulamalar geliştirmek için oluşturulan bir Java altyapısıdır. Birçok grafik araç ile hata ayıklama ve test için güçlü bir destek sağlar. FIPA standartlarına uygun olarak çalışan altyapı, PDA ve mobil cihazlar için de küçük bileşenler eklenerek çalışacak uygulamalar geliştirmeye olanak sağlamaktadır.

Jadex

Amaca yönelik etmenler geliştirmek üzere oluşturulan bir yazılım altyapısıdır. Java ve XML gibi iyi bilinene teknolojilerle etmen geliştirmek üzere tasarlanmıştır.

DESIRE (DEsign and Specification of Interacting REasoning components)

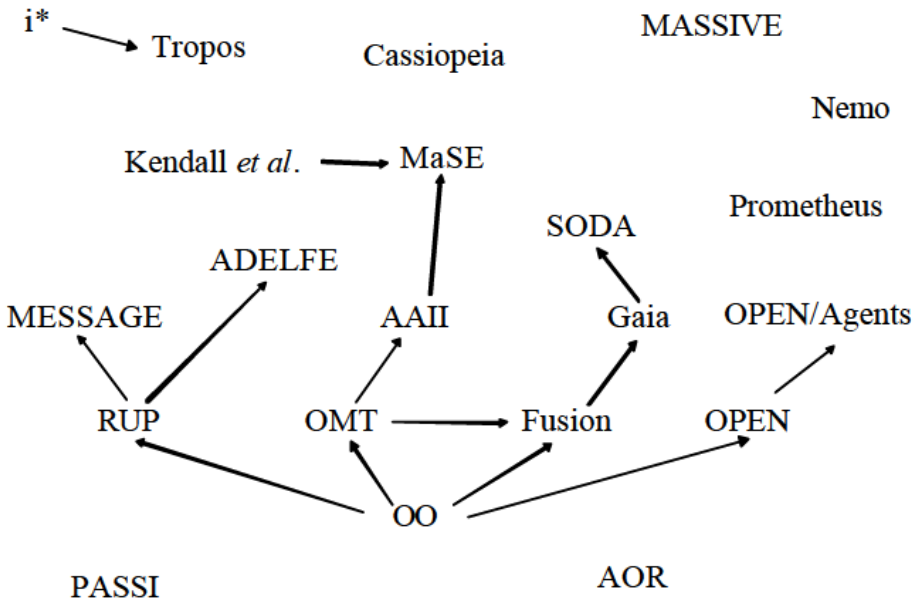
Bu altyapı işlem bileşeni ve bilgi bileşeni ve bu iki bileşen arasındaki ilişkiye dayanır. Bu bileşen temelli yaklaşımda birçok işlem bileşeni birbirine bağlı ya da ayrı olarak çalışır ve çalışmaları bilgi bileşenlerine bağlıdır.

5.3.6. Metodolojiler

Metodoloji bir işin yapılış biçimini tanımlar. Etmene yönelik programlama, yeni bir yaklaşım olduğu için kendine has metodolojilerle geliştirilmek durumundadır. Doğal olarak kendinden önceki nesneye yönelik programlama ve yapısal programlama temelleri üzerine kurulmuş

olan etmene yönelik programlama metodolojilerinden bazıları da nesneye yönelik programlama metodolojilerinin üzerine kurgulanmıştır.

Sistem gereksinimlerinin belirlenmesinden uygulamanın çalışır hale gelmesine kadar geçen sürelerde hangi adımların olacağı ve bu adımların nasıl atılacağına yönelik işlemler, etmene yönelik programlamada şekil 4-1 de görüleceği üzere birçok farklı metodolojiyle yapılabilir.

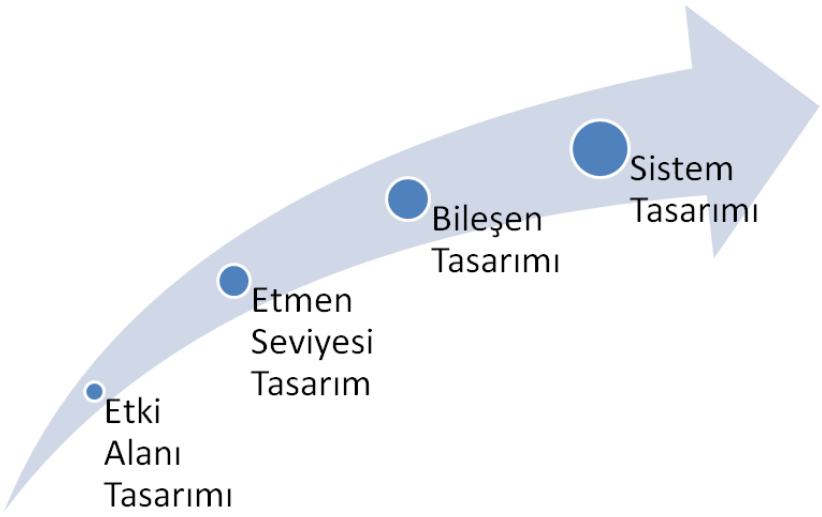


Şekil 5-1 Etmene Yönelik Yazılım Geliştirme Metodolojileri [29]

Bu başlık altında, etmene yönelik yazılım geliştirirken izlenmesi gereken metodolojilerden üç tanesi örnek olarak incelenmiştir.

MaSE

MaSE (Multiagent Systems Engineering) etmene yönelik uygulama tasarlayacak ve geliştirecek olanlar için tam bir metodoloji sağlamaktadır. Metodoloji sistem için gerekli olan belirtimin tanımlanmasından sistemin uygulamasına kadar tüm süreci detaylı olarak tanımlamıştır. Sürecin dört fazı; etki alanı seviyesi tasarım, etmen seviyesi tasarım, bileşen tasarımı ve sistem tasarımı şeklindedir.



Şekil 5-2 MaSE Fazları

MaSE uygulamanın hangi platform ya da dille geliştireceği konusunda herhangi bir yönlendirme yapmaz. Sadece tasarım ve geliştirme için yol gösterici niteliktedir.

Prometheus

Prometheus, uzman olmayanlara yönelik tam bir etmene yönelik yazılım geliştirme mühendisliği metodolojisidir. Üç temel fazdan oluşur: sistem belirtiminin oluşturulması, mimari tasarım ve detaylı tasarım. Belirtim oluşturma fazında sistemin çalışacağı ortam ve sitemin amaçları belirlenir. Mimari tasarım aşamasında etmen tipleri belirlenir, genel sistem yapısı tanımlanır ve etmenleri arasındaki iletişim ve koordinasyon tasarlanır. Son olarak detaylı tasarım aşamasında her bir etmenin içsel olarak neler yapacağı ve bunu nasıl gerçekleştireceği belirlenir. Prometheus metodolojisini kullanan iki araç vardır: JACK Development Environment ve Prometheus Design Tool (PDT).

Tropos

Kanada ve İtalya üniversitelerinden bir grup yazar tarafından geliştirilmiş bir etmene yönelik programlama metodolojisidir. Diğer metodolojilerden farklı olarak gereksinim analizi kısmına daha fazla ağırlık verir. Metodoloji beş fazdan oluşmaktadır.

1. İlk gereksinimleri belirleme
2. Kapsamlı gereksinim belirleme

3. Mimari tasarım
4. Detaylı tasarım
5. Geliştirme

The Gaia Methodology

Wooldridge, Jennigs ve Kinny tarafından geliştirilmiştir. Etmene yönelik programlamada analizin ve tasarımın nasıl yapılacağı belirten bir metodolojidir. Gaia ile hem mikro seviyede etmene yönelik analiz ve tasarım yapılabileceği gibi aynı zamanda daha geniş bir bakış açısından etmenler ve aralarındaki ilişkiler de analiz edilerek tasarlanabilir. Gaia'nın analiz aşamasında roller ve roller arasındaki etkileşimler belirlenir. Rollerin sorumlulukları, yetkileri, aktiviteleri ve protokolleri bulunur. Tasarım aşamasında rollerle etmen tipleri eşleştirilir. Ardından servis modeli oluşturulur.

ETMENE YÖNELİK PROGRAMLAMA İLE NESNEYE YÖNELİK PROGRAMLAMANIN KARŞILAŞTIRILMASI

Etmene yönelik programlama (EYP) ve nesneye yönelik programlama (NYP) hakkında önceki başlıklarda verilen bilgilerden sonra, bu başlığın amacı iki programlama yaklaşımını karşılaştırmaktır.

Kronolojik açıdan NYP yaklaşımı EYP'den önce gelmektedir. EYP doğal olarak NYP de dâhil olmak üzere kendinden önce gelen programlama paradigmalarından ve teknolojilerden etkilenmiştir. Bir EYP uygulaması geliştirmek için özel olarak EYP uygulamaları oluşturmak için geliştirilmiş bir dil kullanılabileceği gibi, NYP destekleyen bir dille de EYP uygulamalar geliştirmek mümkündür.

NYP ile EYP nin ilişkisine genel olarak bakıldığında, EYP uygulamalarını kendi başlarına çalışabilir ve belirli amaçları olan NYP uygulamaları olarak nitelemek mümkündür. Kullanıcı etkileşimine ya da dışarıdan bir müdehaleye gerek duymaksızın çalışmak üzere tasarlanmış, belirli bir amacı olan, bu amacına ulaşmak için çeşitli planlara sahip, etrafını algılayıcılarıyla dinleyen ve duruma göre etkinlik gösteren bir NYP uygulaması geliştirildiğinde aslında bir EYP uygulaması geliştirilmiş olunur.

Diğer taraftan etmene yönelik programlama tüm yazılım sorunları için mükemmel tek çözüm değildir. Aslında aynı şey diğer programlama paradigmaları için de geçerlidir. En uygun çözümü geliştirmeniz için doğru olan, Problemin büyüklüğüne, karmaşıklığına ve tipine göre en uygun programlama yaklaşımını seçmektir. Matematiksel bir ispat için mantıksal programlama, tek bir işlevi yerine getirmek için prosedürel programlama, bir firma için gerekli otomasyon için nesneye yönelik programlama ve programlandığı görevi yerine getirmesi gereken bir insansız uçak için de etmene yönelik programlama en uygun yaklaşım olacaktır.

Aşağıdaki başlıklarda etmen ve nesne ile EYP ve NYP kavramları aktiflik/pasiflik, iletişim/mesajlaşma ve kontrol/çalışma başlıklarıyla ele alınmıştır.

5.4. Aktiflik / Pasiflik

NYP'de temel öğe "nesne"dir. EYP'de ise etmendir. Nesneler gerçek yaşamdaki pasif varlıkların soyutlanması için çok uygundur. Örneğin gerçek yaşamdaki otomobil pasif bir öğedir. Ancak bir insan tarafından kullanıldığında harekete geçer ve çeşitli fonksiyonları çalışır. Bu tip bir öğe için nesne kullanılır. Nesneler ancak dışarıdan gelen etkilere tepki gösterirler. Diğer taraftan kendi başına eyleme geçebilen aktif varlıklar için etmenler çok daha uygun olacaktır. Örneğin bir web sitesi arama motoru, belirli aralıklarla interneti tarayarak değişen ve yeni oluşturulan sayfaları indeksler. Bunun için dışarıdan bir etkinin gelmesini beklemez.

Nesneler pasiftir. Dışarıdan gelen etkilere ve çağrılara cevap verir fakat kendi başlarına belirli bir amaca ulaşmak için harekete geçemez, eylem gerçekleştiremezler. Etmenler ise nesneler gibi dışarıdan gelen çeşitli etkilere tepki vermekle kalmazlar, ayrıca kendileri de harekete geçebilirler. Ayrıca etmenler dışarıdan gelen etkilere nasıl tepki vereceklerini iç ve dış durumlarına bakarak karar verebilirler.

Bu bağlamda etmenlere aktif nesneler denilebilir. Diğer bir deyişle bir etmeni, amaçları olan ve bu amaçlar doğrultusunda kendi başına hareket edebilen ve yüksek seviyeli bir dille iletişim kurabilen nesneler olarak kabul edilebilir.

5.5. İletişim/Mesajlaşma

Nesneler nesnenin yapısına sıkı olarak bağlı bir biçimde mesajlaşmaya gerek duyarlar. Bir nesneye gönderilen mesaj, aslında temel olarak ilgili nesneden bir iş yapmasını istemek anlamına gelir. İş isteğini alan nesnenin, gelen istek karşısında ne yapacağı önceden belirlenmiştir. Nesnelerle iletişim kurmak için ara birimlerden faydalanılır.

Etmenler tipik olarak diğer etmenlerin de kullanabileceği/anlayabileceği yüksek seviyeli bir dil kullanarak iletişim kurarlar. Etmenlerle iletişim kurmak için yüksek seviyeli bir dil kullanarak eşleyici etmenlerden faydalanılır.

5.6. Kontrol/Çalışma

Etmenler de nesnelere benzerler. Fakat onlar nesnelerden farklı olarak fikir, amaç, niyet gibi mental tavırları da desteklerler.

Nesnelerin dışarıdan kontrol edilmesine rağmen, etmenler kendi kendilerini kontrol edebilirler.

Geleneksel nesneye yönelik programlar tek bir iş parçacığı üzerinden çalışır. Etmenlerse eş-zamanlı çalışan öğelerdir.

Etmenler nesnelerden farklı olarak birlikte çalışabilir, rekabet edebilir, uzlaşabilir.

Etmenlerin nesnelere göre daha yüksek yeniden kullanılabilirlik oranına sahiptir. Ayrıca etmenlerin soyutlama düzeyleri de daha kapsamlıdır.

Nesneye yönelik programlamada hangi eylemin yapılacağı ve bir eylemin yapılıp yapılmayacağı, nesneyi çağıran tarafından belirlenir. Ancak etmenler çevrelerindeki değişime, bilgi dağarcığına ve planlarına göre hangi eylemleri gerçekleştireceklerine kendileri karar verirler.

Etmenler diğer etmenlere neyi yapmaları gerektiğini nasıl yapılacağını belirtmeden söyler. Nesnelerdeyse, neyin yapılması gerektiğini nasıl yapılacağını belirterek söylemek gerekir. Bu yapılacak işlemin sabit olarak kodlandığı belirli bir metodu çağırmak anlamına gelmektedir.

Nesneler içlerinde bulundukları ortamla olaylar (events) aracılığıyla iletişim kurarlar. Ancak nesnelerin olayları algılayabilmeleri için, uygun şekilde programlanmış olmaları gerekir. Etmenlerse etraflarını bünyelerinde var olan algılayıcılarla algılarlar ve bu algılama sonrasında ne yapacaklarına yine aynı ortamı etkileyecek tepkilerde bulunurlar.

Nesneler sadece oluşturuldukları zaman belirlenen işlemleri uygulayabilirler. Etmenler de nesneler gibi mesaj iletimiyle çalışırlar. Ancak etmenler nesnelerden farklı olarak bu mesajlara farklı durumlara göre farklı tepkiler verebilirler. Etmenlerin mesajlara farklı cevaplar vermesini sağlayan şey öğrenebilir olmalarıdır.

EYP, NYP den farklı olarak mental durum tavırları taşıdığından fikir, yetenek ve karar gibi özelliklere sahiptir. Etmenler bilgilenme, talepte bulunma, öneride bulunma, kabul etme, red-detme ve diğer etmenlere yardım etme gibi işlemler gerçekleştirebilirler.

Etmenler de nesneler gibi durum bilgisi ve davranışları içerir (encapsulate). Ama etmen bir nesneden fazla olarak akılcı karar verme sistemidir.

Nesneler durumlarını diğer nesnelerin erişebileceği özellikler (attributes) ya da gizlenmiş bilgiler olarak tutarken, etmenler oluşturulurken belirlenen bilgiler, etraflarından algıladıkları, diğer etmenlerden gelen bilgiler ve kendi yaptıkları eylemler de dahil, sürekli gelişen bir bilgi tabanı tutarlar.

Nesneler belirli bir işlem (process) üzerinde çalışırken, etmenler (özellikle dağıtık-etmenler) FIPA standartlarına uygun olarak geliştirilmiş dağıtık etmen platformları üzerinde çalışmaktadırlar.

Nesneler dinamik davranışlar sergileyebilirler. Ancak bu davranışların dışarıdan tetiklenmesi gerekmektedir. Etmenlerse kendi başlarına hareket edebilirler.

6. ÖRNEK UYGULAMA: AKTİF BELGE İNDEKSLEME SİSTEMİ

Örnek uygulamanın amacı etmene yönelik programlamanın temel ilkelerini esas alan bir uygulamanın nasıl modellendiğini ve hayata geçirildiğini göstermektir. Uygulama özerk ve çoklu-etmen yaklaşımları üzerine inşa edilmiştir ve kullanıcıyla etkileşim için bir Windows arayüzüne sahip sahiptir.

Aktif Belge İndeksleme Sistemi (ABİS), birçok bilgisayardan oluşan bir ağ üzerinde hizmet verebilecek ve çeşitli dosya tipindeki belgeleri indeksleyerek herhangi bir kullanıcı tarafından belgenin gerçek yerini bilmeye gerek kalmaksızın kullanılmasını sağlayacak bir sistemin prototipidir.

Prototipin amaca, kullanıcının Acrobat PDF formatındaki dokümanlarının fiziksel konumlarını değiştirmeksizin belirli anahtar kelimeler üzerinden indekslenmesinin sağlanmasıdır. Böylece kullanıcı bilgisayarına yeni bir dosya indirdiğinde ya da kopyaladığında sistem otomatik olarak bu dosyanın içeriğini tarayacak ve içerikte mevcut olan anahtar kelimelere bakarak, kullanıcının tanımladığı anahtar kelimelere göre dosyayı indeksleyecektir. Ayrıca disk üzerindeki belgelerin taşınması, silinmesi ya da isimlerinin değiştirilmesi durumu da ABİS tarafından sürekli olarak izlenecek ve kullanıcının herhangi bir zamanda belgenin son haline ulaşması sağlanacaktır. Kullanıcı arayüzden tanımlamış olduğu herhangi bir indeksi seçtiğinde bu indekse bağlı bütün belgeler kendisine listelenecek ve kullanıcı ABİS üzerinden bu belgeleri çalıştırarak okuyabilecektir.

Sistem bir bilgisayara kurularak çalışmaya başlatıldığında ilk olarak kullanıcıdan sanal klasörler (indeksler) oluşturmasını isteyecektir. Kullanıcı, oluşturduğu her bir indeks (sanal klasör) için belirli anahtar kelimeler tanımlar. Etmenler bu anahtar kelimeleri doküman içinde tarayarak bulunması halinde ilgili belgeyi indeksler.

ABİS ilk kurulduğunda içinde bulunduğu ortamdaki tüm diskleri indeksleyecektir. Sonrasında da disk üzerindeki faaliyetleri sürekli olarak takip ederek, dosyaların fiziksel konum değişikliklerini indekste günceller. Ayrıca bilgisayara eklenen her yeni belge hemen indekslenecektir.

6.1. Uygulamada Kullanılan Teknoloji, Programlama Dili ve Araçlar

Uygulamada temel olarak Microsoft Windows işletim sistemi üzerinde uygulama geliştirme üzerine uzmanlaşmış aşağıdaki teknoloji, programlama dili ve araçlar kullanılmıştır.

6.1.1. Microsoft.NET Altyapısı

Uygulamada platform olarak Microsoft.NET altyapısı kullanılmıştır. .NET alt yapısı dosya erişimi, veritabanı erişimi, disk yönetimi gibi işlemler için tamamen nesneye yönelik olan çok geniş ve kullanışlı bir kitaplığa sahiptir. Bu kitaplığın sağladığı hızlı ve kolay programlama avantajı kullanılmıştır.

6.1.2. Microsoft Visual Studio 2008

Programlama ortamı olarak Visual Studio 2008 Professional yazılım geliştirme ortamı kullanılmıştır. Visual Studio, Microsoft'un Visual Basic ve Visual C# dilleriyle hızlı uygulama geliştirme (RAD - Rapid Application Development)esasına dayalı olarak çalışan ve Windows Form, mobil, web ve servis uygulamaları geliştirmek amacıyla kullanılmıştır.

6.1.3. Microsoft Visual C# Programlama Dili

Microsoft'un .NET altyapısı üzerende nesneye yönelik programlar geliştirmek için geliştirdiği programlama dilidir. Sözdizimi olarak Java ve C'ye yakınlığı nedeniyle tercih edilmiştir.

6.1.4. Microsoft SQL Server 2005

Uygulamayla ilgili veri yönetimi Microsoft SQL Server 2005 üzerinde gerçekleştirilmiştir. İndekslerin, sanal klasörlerin ve genel parametrelerin saklanması ve yönetimi Microsoft SQL Server 2005 ile geliştirilen veritabanı, tablo ve yerleşik yordamlarla sağlanmıştır.

6.1.5. XML Web Servisleri

Etmenlerin haberleşmeleri içinse XML Web Servisleri teknolojisi kullanılmıştır. XML Web Servisi ASP.NET teknolojisi ile geliştirilmiştir.

6.1.6. Microsoft Internet Bilgi Servisleri (Internet Information Services)

Geliştirilen XML Web Servisleri sunucusu olarak, Microsoft Internet Information Services kullanılmıştır.

6.2. Örnek Uygulamanın Gereksinimleri

Örnek uygulamanın gereksinimleri işlevsel ve işlevsel olmayan gereksinimler olarak aşağıdaki şekilde belirlenmiştir:

6.2.1. İşlevsel Gereksinimler

1. Kullanıcı uygulama arayüzünden istediği sayıda ve seviyede sanal klasörler yaratabilecek ve bu klasörler için dilediği kadar anahtar kelime (indeksleme kriteri) girebilecektir.
2. Uygulama kullanıcının bilgisayarındaki metin içerikli tüm Acrobat PDF dokümanlarını tarayacak ve kullanıcının tanımlamış olduğu anahtar kelimelere göre indeksleyecektir.
3. Uygulama internet üzerinden indirme ya da herhangi bir depolama biriminden kopyalama yoluyla bilgisayara yeni kayıt edilen Acrobat PDF dosyalarını otomatik olarak algılayacak ve indeksleyecektir.
4. Uygulama indekslenmiş olan Acrobat PDF dosyalarının fiziksel konumlarını sürekli takip edecek, herhangi bir değişiklik olması durumunda indeks üzerinde dosyanın değişen fiziksel konum ya da isim bilgilerini güncelleyecektir.
5. Kullanıcı indeksleme kriterleriyle ilgili herhangi bir değişiklik yaptığında, uygulama indekslemeyi yenileyecektir.
6. Kullanıcı, uygulama arayüzünden anahtar kelime/kelimeler kullanarak arama yapabilecek ve bu arama kriterlerine uyan dosyalar listelenerek, gerektiğinde arayüz üzerinden Acrobat PDF dosyaların açılması sağlanacaktır.
7. Kullanıcı indekslenemeyen Acrobat PDF dosyalarının listesini görüntüleyebilecektir.

6.2.2. İşlevsel Olmayan Gereksinimler

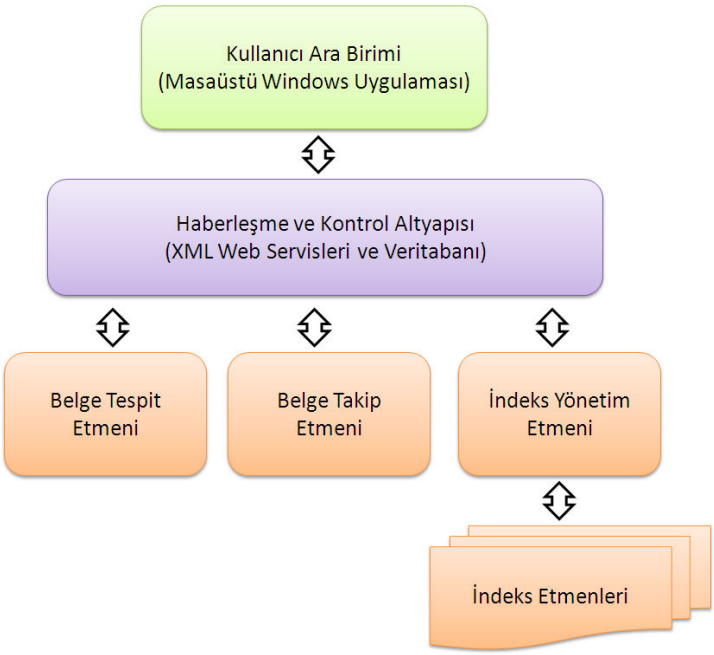
1. İndeksleme işlemi arka planda gerçekleşecek ve işlemci performansı parametrelerde belirtilen değerin altında olduğunda çalışacaktır. İndeksleme çalışırken sistemin genel performansı etkilenmemelidir.
2. Kullanıcı arabirimi 'Windows Forms" uygulaması olmalıdır.

6.2.3. İstisnalar ve Kapsam Dışı Konular

1. İçeriği görsel (imaj) olan Acrobat PDF dosyaları ve diğer belge türleri (.txt, .doc, vb.) indeksleme işlemi dışında tutulmuştur. (madde 7.2.1)

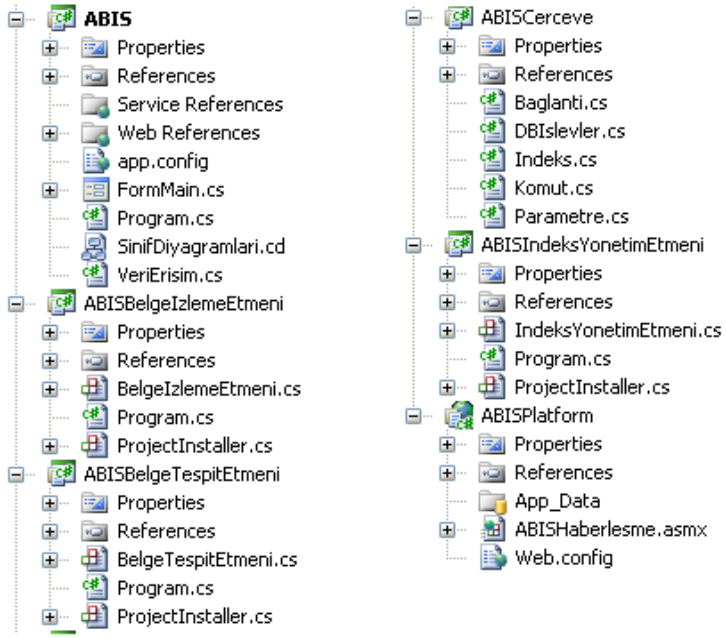
6.3. Çözüm Tasarımı

Bu çalışmanın konusu gereği, bahsedilen gereksinimlerin etmene yönelik programlamayla çözümlenmiştir. Çözümün en ön ucunda, kullanıcının indeksleri oluşturacağı ve yöneteceği bir masa üstü Windows uygulaması oluşturulmuştur. Kullanıcıyla etkileşimi gerektiren işler bu arayüz aracılığıyla yürütülmektedir. Diğer tüm iş ve işlemleri için birlikte çalışan üç ana etmen tasarlanmıştır. Ayrıca etmenlerin kontrolünü ve birbirleriyle iletişimini sağlayan bir de altyapı oluşturulmuştur.



Şekil 6-1 Uygulamanın Temel Mimari Yapısı

Sistemin tamamı için 2 Windows Forms, 3 Windows Service, 1 ASP.NET XML Web Service ve bir de Class Library tipinde proje oluşturulmuştur. Projelerle ilgili detaylara aşağıdaki başlıklarda yer verilmiştir.



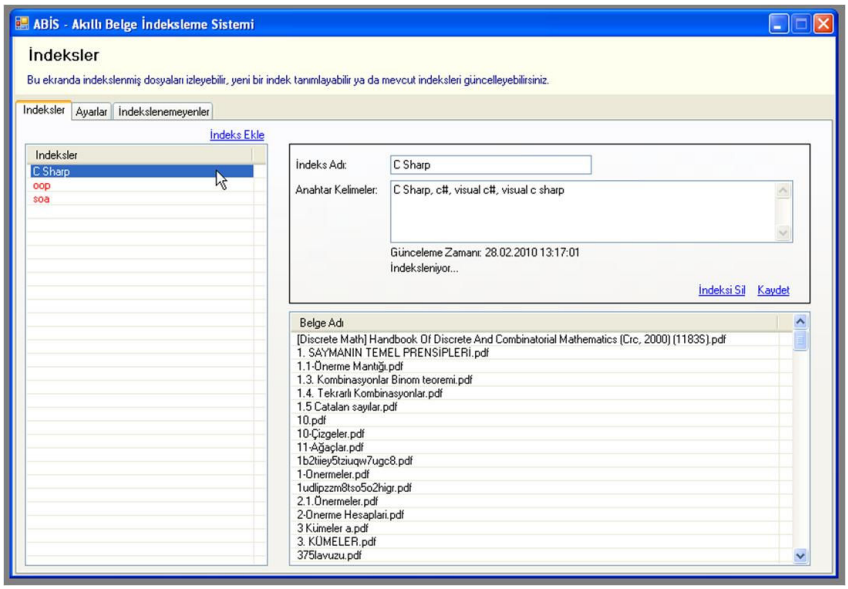
Şekil 6-2 Uygulama İçin Oluşturulan Visual Studio Projeleri

Tüm etmenler kendi prosesleri içerisinde, birbirlerinden bağımsız ancak birlikte çalışacak şekilde tasarlanmışlardır. Etmenlerin her biri bilgisayar çalıştığı anda devreye girer ve normal şartlarda bilgisayar kapanana kadar çalışmalarına devam ederler. Her bir etmen belirli bir amacı vardır ve bu amacı gerçekleştirmek için bir ya da daha fazla sayıda plana sahiptir. Etmenler çevrelerini algılamalarını sağlayan çeşitli algılayıcılara sahiptirler. Bu algılamalara bağlı olarak tepki gösterirler. Ayrıca her etmen proaktif olarak kendi amacını gerçekleştirmek üzere çevresindeki uyaranlardan bağımsız olarak da harekete geçebilmektedir.

6.3.1. Windows Kullanıcı Ara Yüzü

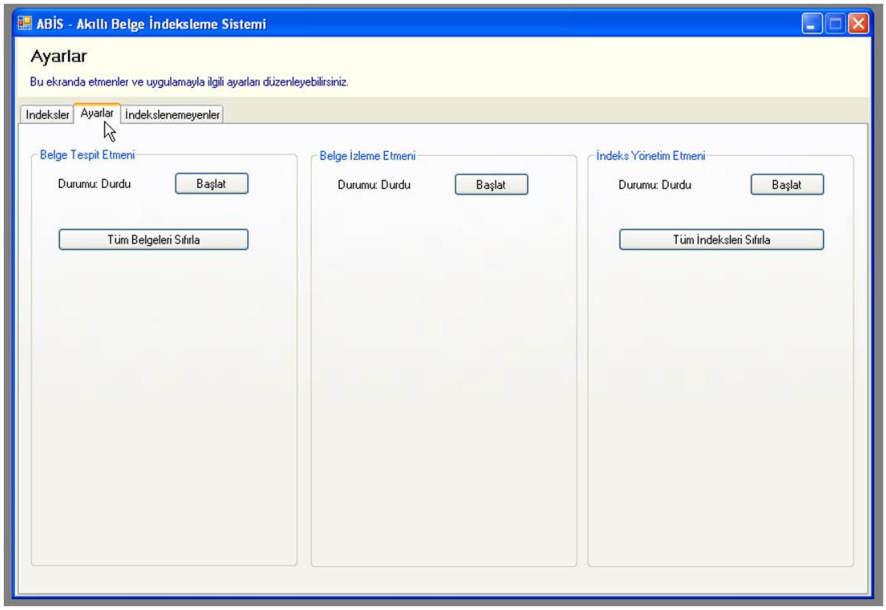
Kullanıcının indeksleri tanımladığı, düzenlediği, indekslerle ilgili anahtar kelimeleri yönettiği ve sistemin geneliyle ilgili ayarları kontrol ettiği uygulama, masa üstü Windows Forms uygulaması olarak geliştirilmiştir.

Uygulamada kullanıcının indeksleri eklediği, sildiği ve güncellediği bir "İndeksler" sekmesi, İndekslerle ilgili ayarlamalar yaptığı bir "Ayarlar" sekmesi ve indekslenemeyen belgelerin listelendiği bir "İndekslenemeyenler" sekmesi bulunmaktadır.



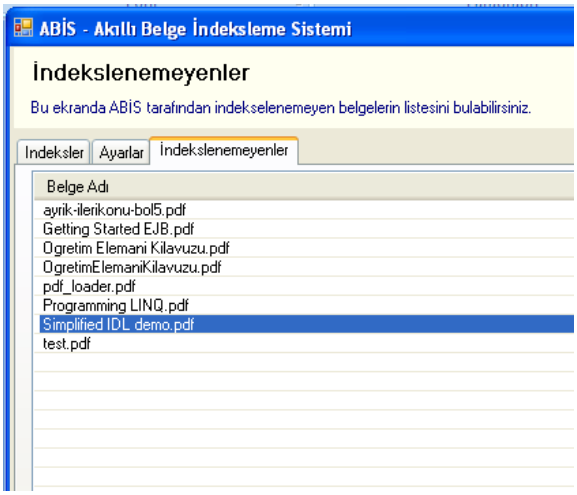
Şekil 6-3 Kullanıcı ara birimi – indeks yönetim sekmesi

Şekil 7.3 de görünen İndeks Yönetim sekmesinde sisteme yeni bir indeks ekleme, mevcut bir indeksle ilgili bilgileri izleme ve güncelleme, indeks silme gibi işlemler yapılabilmektedir. Ayrıca eğer ilgili indeksle indekslenmiş olan belge varsa, kullanıcı belgeler listesinde bu belgeleri görebilmekte ve çift tıklayarak belgenin açılmasını sağlayabilmektedir



Şekil 6-4 Kullanıcı ara birimi – etmen yönetim sekmesi

Şekil 7.4 de ise, etmenlerin durumlarını gösteren ve yönetilmelerini sağlayan sekmeyi görmektesiniz. Kullanıcı bu sekmeyi kullanarak etmenleri durdurabilir ya da başlatabilir. Ayrıca mevcut belge ve indeks bilgilerini sıfırlayarak, belge tespit ve indeksleme işlemlerini sıfırdan başlatmak mümkündür.



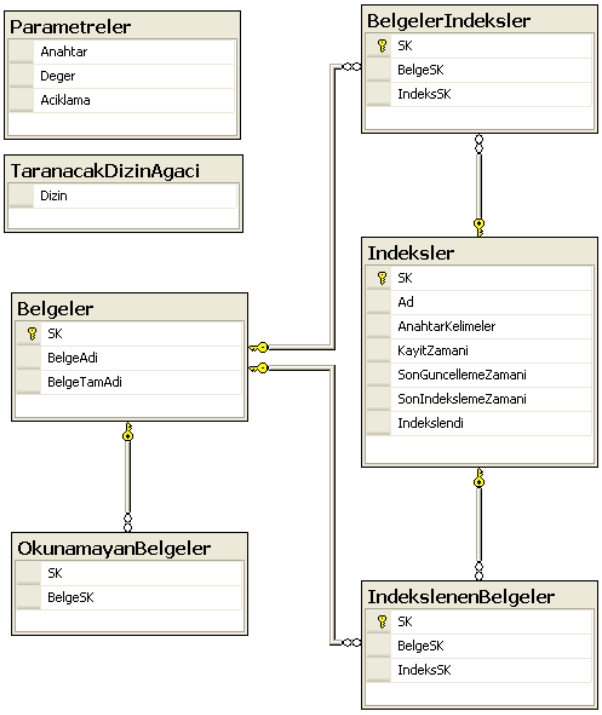
Şekil 6-5 Kullanıcı ara birimi – indekslenemeyen belgeler

Şekil 7.4 de görünün İndekslenemeyen belgelerse, kilitli, imaj biçiminde olan ve diğer sebeplerden sistem tarafından metin olarak okunamayan PDF dosyaların listelendiği ekrandır. Kullanıcı bu ekranda da istediği bir dosyayı çift tıklayarak açabilmektedir.

Arayüz, yeni bir indeks eklendiğinde ya da mevcut bir indeks güncellendiğinde ABİS Haberleşme Servisine mesaj yollayarak durum haber verir. Haberleşme servisi de durumu ABİS İndeks Yönetim Etmenine ileterek, gerekli indeksleme işlemlerini başlatmasını sağlar.

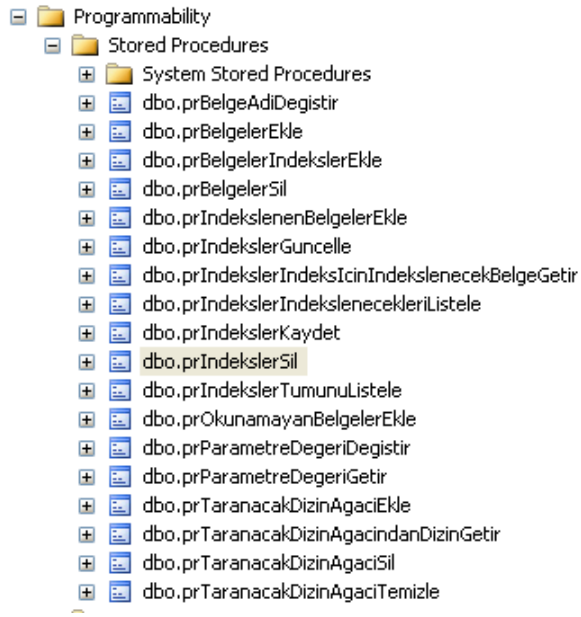
6.3.2. ABİS Veritabanı

SQL Server 2005 üzerine yapılandırılmış olan ABIS veritabanı, etmenlerin durumlarını ve yaptıkları işlerle ilgili bilgileri ve sistemin asıl işi olan indeks bilgilerini depolamak; eklemek, güncellemek, silmek ve listelemek amacıyla kullanılmıştır.



Şekil 6-6 Uygulamanın Veritabanı Yapısı

Arayüz ve etmenlerin tüm veritabanı işlemleri, veritabanı içerisindeki yerleşik yordamlar (stored procedure) aracılığıyla sağlanmaktadır.



Şekil 6-7 Veritabanı Operasyonları İçin Kullanılan Yerleşik Yordamlar

6.3.3. ABİS Çerçeve

Veritabanı bağlantısı açma, komut nesnesi oluşturma, veritabanı komut parametreleri oluşturma, bunlara bağlı olarak veritabanındaki değerleri okuma, listeleme, güncelleme gibi iş ve işlemler için Windows arayüzü ve etmenlerce ortak kullanılacak bir dizi nesne ve bu nesneler içinde nitelik ve fonksiyonlar tanımlanmıştır.

6.3.4. ABİS Belge Tespit Etmeni

Belge tespit etmeni uygulama bir bilgisayara ilk defa kurulduğunda ya da mevcut belge bilgileri temizlendiğide çalışır. Etmen önceklikle bilgisayarın disklerinin ve bu diskler içindeki dizinlerin bir haritasını çıkartır. Daha sonra bu haritayı kullanarak, içinde PDF dosya olan her bir klasördeki belgeleri tarayarak, bilgisayardaki PDF belgelerinin listesini çıkartarak veritabanına aktarır.

Etmenin sağlamlığını (robustness) sağlayan, her adımın etmenin durum bilgisi olarak kaydedilmesidir. Bilgisayarın belirli aralıklarla kapatılıp açılacağı düşünüldüğünde, sistemin kararlı ve her seferinde yeniden başlamaya gerek duymadan çalışması için, yapılan her işlem kaydedilmektedir.

Etmenin birincil amacı bilgisayar içerisindeki PDF belgelerini tespit etmek ve indekslenmek üzere veritabanına kaydetmektir. Bu amacı gerçekleştirmek için belirlediği alt amaçlar aşağıdaki gibidir.

1. Dizinlerin belirlenmesi ve veritabanına kaydedilmesi
2. Her bir dizindeki PDF belgelerin belirlenerek veritabanına kaydedilmesi

Etmen bu amaçlarına ulaşabilmek için aşağıdaki plan ve adımları uygulayabilmektedir.

Dizinlerin belirlenmesi ve veritabanına kaydedilmesi için;

1. Bilgisayardaki diskleri bul
2. Her bir disk içindeki dizinleri bul
3. İçinde PDF belge olan dizinlerin listesini oluştur
4. Dizin listesi tamamlanınca tamamını veritabanına kaydet
5. Buraya kadar olan adımlarda bir sorun varsa baştan (1 nolu madde) başla

Her bir dizindeki PDF belgelerin belirlenerek veritabanına kaydedilmesi için;

1. Veritabanındaki her bir dizinin içindeki PDF belgeleri listele
2. PDF belge listesini veritabanına kaydet
3. Dizini tespit edilen dizini veritabanından sil

Etmenin proaktifliğini, belirli zaman aralıklarıyla kendi durumunu ve ABİS veritabanını kontrol etmesi sağlamaktadır. Etmen ayrıca üzerinde çalıştığı sistemin yükünü artırmamak için işlemci kullanım algılayıcısı kullanmaktadır.

6.3.5. ABİS Belge İzleme Etmeni

Bilgisayar içindeki belgeler bir kez tespit edildikten sonra yeni eklenecek belgelerin, silinecek belgelerin, belge isim değişikliklerinin ve belgelerin yer değişimlerinin indekslemenin sağlıklı olabilmesi için izlenmesi gerekmektedir. Belge izleme etmeni, sistem bilgisayara kurulduğundan itibaren devreye girer ve bilgisayar açık olduğu sürece çalışır. Bu süreç içerisinde işletim sistemini dinleyerek her türlü dosya işlemini takip eder. Kes-yapıştır, dosya silme, dosya ismi değiştirme gibi işlemlerin her biri için belge sisteme kayıtlıysa gerekli takibi ve güncellemeleri yapar.

Etmenin birincil amacı bilgisayar üzerindeki PDF belgelerdeki taşıma, silme, kopyalama, isim değiştirme gibi değişimlerin takip edilerek, dosyanın yolu ve adının veritabanında güncellen-

mesidir. Etmen bu amacı gerçekleştirmek için dosya değişiklikleri algılayıcısına sahiptir. Bu algılayıcı sadece PDF dosyalar için çalışır ve değişiklikleri etmene bildirir. Etmen de bu bilgilere göre değişikliği uygun bir şekilde yansıtabilmek için aşağıdaki plan ve adımları uygular.

- Değişiklik: Belge taşıma
  1. Belgenin önceki adını ve yerini tespit et
  2. Belgenin yeni adını ve yerini tespit et
  3. Belgenin eski yerini veritabanında bul ve ilgili belge kaydını yeni belge adı ve yeriyle güncelle

- Değişiklik: Belge silme
  1. Silinen belgenin adını ve yerini tespit et
  2. Belgenin eski yerini kullanarak veritabanı kaydını bul ve sil
  3. Belgeyle ilgili indeksleme bilgilerini veritabanından sil

- Değişiklik: Yeni belge ekleme
  1. Yeni belgenin adını ve yerini tespit et
  2. Yeni belgeyi veritabanına ekle
  3. İndeks Yönetim Etmenine, indekslenecek yeni belgeyi haber ver.

- Değişiklik: Dış kaynaktan (CD, DVD, internet, vb.) belge kopyalama
  1. Yeni belgenin adını ve yerini tespit et
  2. Yeni belgeyi veritabanına ekle
  3. İndeks Yönetim Etmenine, indekslenecek yeni belgeyi haber ver.

6.3.6. ABİS İndeks Yönetim Etmeni

Bilgisayardaki belgelerle kullanıcının tanımlamış olduğu indeksleri ilişkilendiren etmen İndeks Yönetim Etmenidir. Emen devreye girdiği anda, öncelikle indekslenmemiş belge olup olmadığını kontrol eder. Daha sonra belge eşleşmesi tamamlanmamış her bir indeks için ayrı birer indeksleme etmeni (yeni birer process) oluşturarak tüm belgelerin indekslenmesini sağlar.

İndeksleme işlemi, her bir indeksin, tüm belgelerin içeriklerini tarayarak, indeks için belirlenmiş olan anahtar kelimeleri içerip içermediğini belirlemesi şeklinde gerçekleşir. Eğer belge indeksin anahtar kelimelerini içeriyorsa, belge ve indeks birbirleriyle ilişkilendirilerek veritabanına kaydedilir.

Etmenin birincil amacı her bir indeks için tüm belgelerin taranarak, varsa eşleşenlerin kaydedilmesidir. Etmen bu amacını gerçekleştirmek için aşağıdaki plan ve adımları her bir indeks için ayrı ayrı uygular.

1. Belgeler listesinden indeksle eşlemesi yapılmamış bir belge al.
2. İndeks için belirlenmiş anahtar kelimelerin belgenin içinde olup olmadığına bak; eğer varsa belge-indeks ilişkisini veritabanına kaydet.
3. Belgeyi indeksle ilişkili olarak indekslenmiş belgeler tablosuna kaydet.
4. Varsa bir sonraki indekslenecek belgeyi al (1 nolu madde) ve buraya kadar olan işlemleri tekrar et. Eğer indekslenecek belge kalmadıysa, ilgili indeksi, indekslenme işlemi tamamlandı olarak işaretle.

Etmen, indeksleme işlemini sistemin genel performansını etkilemeden yapmak için zamanlayıcı ve işlemci kullanım oranı algılayıcılarını kullanarak gerekli ayarlamaları yapmaktadır.

6.3.7. ABİS İndeksleme Etmeni

İndeks yönetim etmeni, her bir indeks için yeni bir işlem (process) içerisinde açar. Bu işlem bir Windows Form uygulamasını baz alır. İndeks yönetim etmeni veritabanını tarayarak indekslenmesi tamamlanmamış olan sanal klasörleri tespit eder ve her bir indeks için yeni bir İndeksleme Etmeni oluşturarak, bu yeni oluşturduğu işleme indeksin veritabanı numarasını (SK) yollar. Böylece indekseleme etmeninin kendisine yolladığı bu parametreyle, ilgili anahtar kelimeler üzerinden tespit edilmiş belgeleri indeksler.

6.3.8. ABİS Haberleşme Servisi

Haberleşme servisi, kullanıcı arabirimi ve etmenlerin birbirleriyle iletişimini sağlamak amacıyla tasarlanmıştır. Böylece prosesler arası veri paylaşımı için yüksek seviyeli bir teknoloji kullanılmıştır.

Sistemde kullanılan haberleşme ilk olarak kullanıcı arabiriminin etmen yönetiminde kendini göstermektedir. İndeksleri sıfırlama, belgeleri sıfırlama gibi işlemler, arayüzün XML Web Servis aracılığıyla etmenlere yolladığı mesajlarla gerçekleştirilmektedir. Ayrıca Belge İzlemi Etmeni, bilgisayara yeni bir belge eklendiğinde, İndeks Yönetim Etmenine mesaj göndererek indeksleme komutu vermektedir.

6.4. Çözümün Uygulanması

Bu başlık altında çözümün bölüm 6.1 de açıklanan teknoloji ve araçlarla bir etmenin nasıl gerçekleştirileceğine değinilecektir. Visual Studio kullanılarak bir Windows işletim sistemi üzerinde çalışacak etmene yönelik programlama yapmak için, Windows Service proje tipi kullanılır.

6.4.1. Çözümün Uygulanması İçin Temel Adımlar

Yeni bir Windows Uygulama oluşturmak için aşağıdaki adımlar uygulanmıştır.

1. Yeni Bir Windows Service uygulaması başlatın. Bunun için menüden File\New\Project seçeneiğini tıklayın. Açılan New Project diyalog penceresinden tercihinize göre Visual Basic ya da C# dilini tercih edin. Ardından proje tipi olarak Windows'u şablonlardan da Windows Service'i seçin.
2. Yeni oluşturduğunuz projedeki servis sınıfı içerisinde, sınıf seviyesinde gerekli değişkenleri tanımlayın. Servis başlamadan hemen önce yapılması gerekenleri yapıcı (constructor) metoduna yazın. Örneğin etmen dosya isim ya da yer değişikliği gibi belirli sistem olaylarıyla çalışacaksa, bununla ilgili tanımlamaları burada yapabilirsiniz. Ya da yapıcı içerisinde bir zamanlayıcı (timer) ayarlanabilir.
3. Servisin başlamasıyla birlikte yapılması gereken işleri OnStart olay işleyicisine yazın. Bu yordam, servisin Windows işletim sistemi üzerinde çalışmaya

başlamasıyla birlikte tetiklenir. Bu yordam içerisinde zamanlayıcı başlatılabilir, sınıf seviyesinde tanımlanan nesneler örneklendirilebilir, veritabanı ve dosya bağlantıları oluşturulabilir.

4. Servisin durduğunda çalışması gereken kodlarsa OnStop olay işleyicisine yazılır. Servisin durması bilgisayarın kapatılması ya da servis yöneticisi aracılığıyla normal yollarla gerçekleştiğinde bu yordam çalışmaktadır. Bu durumda zamanlayıcı, sistem olay işleyicileri ya da veritabanı/dosya bağlantıları gibi sistem kaynağı tüketen nesne ve bileşenler uygun bir şekilde sonlandırılmalıdır.

5. Yukarıdaki temel adımlar atıldıktan sonra etmenin temel işlevleri kodlanır. Bu işlevler olay işleyicilere ya da zamana bağlı olarak çalıştırılır. Etmenin sağlamlığını (robustness) sağlamak için yapılan her işlem bellek haricinde bir yere (veritabanı/disk) yazılır. Böylece etmen yeniden başladığında durum bilgisini yükleyerek stabil bir noktadan çalışmasına devam edebilecektir.

6. Windows Service'in bilgisayara kurulabilmesi için oluşturulan servis için bir de ProjectInstaller bileşeni eklenir. Bunun için Visual Studi'da ilgili Windows Servisinin üzerine sağ tıklayarak Add Installer demek yeterlidir.

7. Oluşturulan servis Visual Studio üzerinden Build komutuyla derlenir ve ortaya çıkan çalıştıralabilir dosya (executable) installutil aracıyla bilgisayara kayıt edilir.

8. Servis yöneticisinden yüklenen yeni servis başlatılıp, durdurularak yönetilebilir.

6.4.2. Belge İzleme Etmeninin Geliştirilmesi

Bu başlıkta ABİS Belge İzleme Etmeninin geliştirilmesi adım adım örnek kodlarla birilkte anlatılmıştır. Kodun tamamı ekler kısmında bulunmaktadır.

1. Yeni bir Windosw Service projesi başlatılarak adı ABISBelgeIzlemeEtmeni olarak verildi.

2. Servisin adı BelgeIzlemeEtmeni olarak değiştirildi.

3. Servisin kodlanması için aşağıdaki using deyimleri proje dosyasına eklendi.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Linq;
using System.ServiceProcess;
using System.Text;

using System.IO;
```

```
using ABISCerceve;
using System.Threading;
```

4. Dosya değişimleri takip etmek amacıyla kullanılacak olan zamanlayıcı, dosya izleyici ve değişen dosya bilgileri için oluşturulan nesneyle ilgili değişkenler sınıf seviyesinde tanımlanır.

```
// izleyiciler listesi için koleksiyon
List<FileSystemWatcher> izleyiciler = new List<FileSystemWatcher>();
System.Timers.Timer zamanlayici;
Queue<DegisenDosyaBilgisi> degisenDosyalar = new
Queue<DegisenDosyaBilgisi>();
```

5. Servisin yapıcı metoduna zamanlayıcıyla ilgili nesne örneklendirme ve olay işleyici atama kodları yazılır.

```
public BelgeIzlemeEtmeni()
{
    InitializeComponent();
    zamanlayici = new System.Timers.Timer(250);
    zamanlayici.Elapsed
        += new System.Timers.ElapsedEventHandler(zamanlayici_Elapsed);
}
```

6. Servisin OnStart olay işleyicisine aşağıdaki kod yazılır. Bu kod zamanlayıcı devreye alır ve etmen için her bir disk üzerinde dosyaları izleyen birer algılayıcı oluşturur.

```
protected override void OnStart(string[] args)
{
    zamanlayici.Interval = 250;
    zamanlayici.Start();

    // LINQ ile bilgisayardaki sabit disklerin listesi alınıyor
    var diskler = from disk in DriveInfo.GetDrives()
                  where disk.DriveType == DriveType.Fixed
                  select new { disk.Name, disk.VolumeLabel };

    izleyiciler.Clear();

    // her bir disk için izleyici tanımlanarak olaylar bağlanıyor
    foreach (var oge in diskler)
    {
        FileSystemWatcher izleyici
            = new FileSystemWatcher(oge.Name);
        izleyici.EnableRaisingEvents = true;
        izleyici.NotifyFilter
            = NotifyFilters.FileName | NotifyFilters.DirectoryName;
        izleyici.Filter = "*.pdf";
        izleyici.IncludeSubdirectories = true;
        izleyici.Changed
            += new FileSystemEventHandler(izleyici_Changed);
        izleyici.Created
            += new FileSystemEventHandler(izleyici_Changed);
```

```
            izleyici.Deleted
                += new FileSystemEventHandler(izleyici_Changed);
            izleyici.Renamed
                += new RenamedEventHandler(izleyici_Renamed);

            izleyiciler.Add(izleyici);
        }
    }
```

7. Dosya izleyicileriyle ilgili Renamed ve Changed olay işleyicileri yazılır. Bunlar silinen, taşınan, kopyalanan ya da adı değişen dosyaları izleyen algılayıcılardır.

```
void izleyici_Renamed(object sender, RenamedEventArgs e)
{
    // belgenin veritabanındaki tam adı değiştiriliyor.
    ABISCerceve.DBIslevler.BelgeAdiDegistir(e.OldFullPath,  e.FullPath,
e.Name);
}

void izleyici_Changed(object sender, FileSystemEventArgs e)
{
    switch (e.ChangeType)
    {
        case WatcherChangeTypes.All:
            break;
        case WatcherChangeTypes.Changed:
            break;
        case WatcherChangeTypes.Created:
            DegisenDosyaBilgisi ddb1
                = new DegisenDosyaBilgisi();
            ddb1.Ad = Path.GetFileName(e.FullPath);
            ddb1.DegisiklikTipi
                = DegisenDosyaBilgisi.DegismeTipi.Olusturuldu;
            ddb1.TamAd = e.FullPath;
            degisenDosyalar.Enqueue(ddb1);
            break;
        case WatcherChangeTypes.Deleted:
            DegisenDosyaBilgisi ddb2
                = new DegisenDosyaBilgisi();
            ddb2.Ad = Path.GetFileName(e.FullPath);
            ddb2.DegisiklikTipi
                = DegisenDosyaBilgisi.DegismeTipi.Silindi;
            ddb2.TamAd = e.FullPath;
            degisenDosyalar.Enqueue(ddb2);
            break;
        case WatcherChangeTypes.Renamed:
            break;
        default:
            break;
    }
}
```

8. Dosya değişimleri işleyecek olan yardımcı yordamlar yazılır.

```
private void IkiliDegisimIsle(object ikiliDegisimBilgisi)
{
    IkiliDegisimBilgisi degisimBilgisi
        = (IkiliDegisimBilgisi)ikiliDegisimBilgisi;
    DegisenDosyaBilgisi ddb1 = degisimBilgisi.Birinci;
    DegisenDosyaBilgisi ddb2 = degisimBilgisi.Ikinci;

    if
(ddb1.DegisiklikTipi==DegisenDosyaBilgisi.DegismeTipi.Olusturuldu
                    &
ddb2.DegisiklikTipi==DegisenDosyaBilgisi.DegismeTipi.Silindi)
    {   // Kes / Yapıştır (Taşıma)
        // Veritabanından dosya tam adı değiştiriliyor
        ABISCerceve.DBIslevler.BelgeAdiDegistir(ddb2.TamAd,
ddb1.TamAd, ddb1.Ad);
    }

    if (ddb1.DegisiklikTipi ==
DegisenDosyaBilgisi.DegismeTipi.Silindi
                    & ddb2.DegisiklikTipi ==
DegisenDosyaBilgisi.DegismeTipi.Olusturuldu)
    {  // Silme ya da geri alma
        if (!ddb1.TamAd.Contains("RECYCLER"))
        {   // Silme işlemi
            // Dosya ve indeks bilgileri veritabanından siliniyor
            ABISCerceve.DBIslevler.BelgeSil(ddb1.TamAd);
        }

        if (!ddb2.TamAd.Contains("RECYCLER"))
        {   // Silinen dosya geri alındı işlemi
            // Dosya, dosyalar listesine ekleniyor
            ABISCerceve.DBIslevler.BelgeEkle(ddb2.TamAd, ddb2.Ad);
        }
    }
}

private void TekliDegisimIsle(object degisenDosyaBilgisi)
{
    // yeni dosya eklendi.
    DegisenDosyaBilgisi degisenDosya =
(DegisenDosyaBilgisi)degisenDosyaBilgisi;
    ABISCerceve.DBIslevler.BelgeEkle(degisenDosya.TamAd,
degisenDosya.Ad);
}
```

9. Servisin OnStop olay işleyicinde zamanlayıcı durdurulur ve olay işleyiciler temizlenir.

```
protected override void OnStop()
{
    zamanlayici.Stop();
    foreach (FileSystemWatcher fsw in izleyiciler)
    {
        fsw.EnableRaisingEvents = false;
        fsw.Dispose();
    }
    izleyiciler.Clear();
    degisenDosyalar.Clear();
}
```

10. Servise sağ tıklanarak yeni bir Project Installer eklenir. Proje derlenir ve komut satırından installutil kullanılarak oluşturulan servis uygulamasının tam yolu ve adı verilerek kurulması sağlanır.

SONUÇ

Bu çalışmada akıllı etmenler ve etmene yönelik programlama konuları hakkında temel bilgiler verilmiş, etmene yönelik programlama en çok tercih edilen programlama yaklaşımı olan nesneye yönelik programlamayla karşılaştırılarak konunun anlatımı desteklenmek istenmiştir. Ayrıca geliştirilen örnek bir uygulamayla, etmene yönelik programlamanın gerçek bir senaryo üzerinden tasarımı ve uygulanışı verilerek, etmene yönelik programlamanın nesneye yönelik programlama üzerinde nasıl çalışacağı gösterilmiştir.

Geliştirilen uygulamada akıllı etmenlerin en temel nitelikleri olan bağımsız çalışabilme, sistem üzerinde sürekli var olma/çalışma, belirli bir amacı ve bu amaca ulaşacak planları olma ve etrafını algılayıcılarla algılamaya yer verilmiştir. Ayrıca örnek uygulama bir çok-etmenli sistem olduğundan, etmenler arasındaki iletişim ve koordinasyon bir XML Web Servisi üzerinden sağlanmıştır.

Etmene yönelik programlama birçok farklı dilde gerçekleştirilebilir ve istenirse akıllı etmenler için hazırlanmış özel platformları kullanmak mümkün olabilmektedir. Bu çalışmada yapılan örnek uygulamada akıllı etmenler ve etmenlerin birlikte çalışmaları hiçbir hazır altyapı kullanılmadan gerçekleştirilmiştir. Uygulama bu haliyle sadece bilgisayardaki PDF belgeleri indekslemektedir. Sonraki sürümlerinde uygulama MS Word, MS Excel, CHM gibi diğer belge türlerini de indeksleyebilecek hale getirilebilir. Ayrıca etmenin indekslemesi bu sürümünde mümkün olduğunca basit tutulmuştur. Fakat indeksleme için yapay zeka kullanılarak hangi belgenin hangi indekse ne oranda yakın olduğu tespit edilerek uygulamanın daha da zenginleştirilmesi mümkün olabilecektir. Sistem daha sonra bir sunucu üzerine kurularak ağdaki tüm bilgisayara etmenler gönderip belgeleri indeksleyip sunucudan anahtar kelimelerle erişilir hale getirilmesi mümkündür.

Çalışmanın sonucunda ulaşılan nokta, proaktif olarak belirli bir kullanıcının yönlendirmesinden bağımsız olarak çalışması gereken uygulamalar için en uygun programlama yaklaşımının etmene yönelik programlama olduğu yönündedir. Sistem üzerinde sürekli çalışan ve kendine verilen görevlere uygun olarak çalışması gereken sistemleri akıllı etmen olarak tasarlamak doğru bir yaklaşım olarak görülmektedir. Ancak kullanıcıyla etkileşimin zorunlu olduğu yerlerde olay-güdümlü ve nesneye yönelik programlama daha uygun ve kaçınılmaz bir tercih

olacaktır. Çünkü etmene yönelik programlama kullanıcıdan bağımsız çalışmaktadır ve bu yaklaşımla kullanıcı arayüzü tasarımı mümkün olmaz. Diğer taraftan geniş kapsamlı sistemler için etmene yönelik ve nesneye yönelik programlama paradigmaları birlikte kullanılarak çözüm geliştirilmesi mümkündür ve bu yaklaşım en optimum çözümlerin geliştirilmesini olanaklı kılmaktadır.

KAYNAKLAR

1. Elmas Çetin, “Yapay Sinir Ağları (Kuram, Mimari, Eğitim, Uygulama)”, Seçkin Yayınevi, ISBN:975-347-612-4, Ankara, 2003
2. Wikipedia, http://en.wikipedia.org/wiki/History_of_artificial_intelligence , 2009
3. Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, NJ: Prentice Hall, ISBN 0-13-790395-2, chpt. 2
4. Nabiyev Vasif, “Yapay Zeka – Problemler, Yöntemler, Algoritma”, Seçkin Yayınevi, ISBN: 975-347-985-9, Ankara, 2005
5. Görz Günther, Nebel Bernhard, “Yapay Zeka (Kunstliche Itelligenz)”, İnkilap Yayıncılık, ISBN:975-10-2405-6, İstanbul, 2005
6. Erdur, R.C. “Yazılım Etmenleri: Genel Kavramlar, Mimariler, Standartlar, Araçlar”, COMPOTEK 2005 Davetli Bildiri, İzmir, Kasım 2005.
7. Kadar B., Monostori L., Szelke E., "An Object-Oriented Framework for Developing Distributed Manufacturing Architectures", Journal of Intelligent Manufacturing, No.9, 1998
8. EURESCOM, “Agent Based Computing, A Booklet For Executives”, P712 and P815. http://www.eurescom.de/~pub-deliverables/P800-series/P815/Booklet/!AGENT_B.PDF
9. Shoham Yoav, Agent oriented programming. Technical report, Stanford University, 1992.
10. Woolridge Micheal, An Introduction to MultiAgent Systems, John Wiley & Sons, Chichester, UK, ISBN:047149691X, http://www.csc.liv.ac.uk/~mjw/pubs/imas/ (2002)
11. Sandeep S, Seminar Report On Agent Oriented Programming, Department of Computer Science and Engineering Sree Narayana Gurukulam College of Engineering, Kolenchery, Kasım 2006
12. Maes, Pattie (1995), "Artificial Life Meets Entertainment: Life like Autonomous Agents," Communications of the ACM, 38, 11, 108-114
13. Hayes-Roth, B. (1995). "An Architecture for Adaptive Intelligent Systems," Artificial Intelligence: Special Issue on Agents and Interactivity, 72, 329-365
14. Franklin Stan, Graesser Art. A taxonomy for autonomous agents. http://www.msci.memphis.edu/ franklin/AgentProg.html, 1996. Proceedings of Third International workshop on Agent Theories.
15. Wikipedia, http://en.wikipedia.org/wiki/Intelligent_agent, 2009

16. Jennings N. R., Wooldridge M., “Applications of intelligent agents”, ISBN: 3-540-63591-2 , 1998
17. Pour Gilda, “Integrating Agent-Oriented Enterprise Software Engineering into Software Engineering Curriculum”, Frontiers in Education, 2002. FIE 2002. 32nd Annual, 6-9 Kasım 2002
18. Wooldridge M. J., Jennings N. R., “Software Engineering with Agents: Pitfalls and Pratfalls”, IEEE Internet Computing, 3(3):20–27, Mayıs/Haziran 1999.
19. Shoham Y., “Agent-oriented programming” Artificial Intelligence, (60) 51-92, 1993.
20. Tveit Amund, “A survey of Agent-Oriented Software Engineering”, Norwegian University of Science and Technology, 8 Mayıs 2001
21. Nicholas R. Jennings, Michael Wooldridge, “Agent-Oriented Software Engineering”, Handbook of Agent Technology, J. Bradshaw (ed.). AAAI/MIT Press, 2000.
22. The Foundation for Intelligent Physical Agents (FIPA), http://www.fipa.org/index.html, 2010
23. Bordini R., Braubach L., Dastani M., Seghrouchni F., Gomez-Sanz J., O’Hare G., Pokahr A., Ricci A., “A Survey of Programming Languages and Platforms for Multi-Agent Systems”, Informatica 30, 2006
24. Wikipedia, “Object Oriented Programming”, http://en.wikipedia.org/wiki/object-oriented_programming, 2009
25. Booch Grady, Maksimchuk Robert A., Engle Michael W., Young Bobbi J., Conallen Jim, Houston Kelli A., “ Object-Oriented Analysis and Design with Applications Third Edition”, ISBN 0-201-89551-X, 2007
26. Stefik Mark, Bobrow Daniel G., “Object-Oriented Programming: Themes and Variations”, AI Magazine Volume 6 Number 4, 1985
27. Alaybeyoğul Ayşegül, Erdur Rıza Cenk, “Yazılım Etmenleri: Açık, Dinamik ve Heterojen Ortamlarda Yazılım Geliştirme İçin Bir Teknoloji”
28. LI Jianxing, MAO Xinjun, SHU Yao ,“An OO-Based Design Model of Software Agent”, International Conference on Parallel and Distributed Computing, Applications and Technologies, 2005
29. Pradghan Lin, Winikoff Michael, Developing Intelligent Agent Systems: A Practical Guide, John Wiley & Sons Ltd., ISBN: 978-0-470-86120-2, England, 2005
30. Federico Bergenti, “A Discussion of Two Major Benefits of Using Agents in Software Development”, ESAW 2002 Revised Papers, LNCS 2577, Springer-Verlag, pp:1-12, 2003.

31. Sarıçiçek İnci, Yüzügüllü Nihat, “Çok Ajanlı Melez Atölye Yönetim Sistemi”, Endüstri Mühendisliği 4, 2003
32. Lind J. Issues in Agent-Oriented Software Engineering. The First International Workshop on Agent- Oriented Software Engineering (AOSE-2000), 2000.
33. Brian Henderson-Sellers, “Agent-based Software Development Methodologies”, OOPSLA 2002 Workshop, 2002

ÖZGEÇMIŞ

Kadir Çamoğlu 1974 İstanbul doğumludur. 1995 yılından bu yana profesyonel olarak çeşitli yazılım projelerinde görev almış, veritabanı ve yazılım geliştirme alanlarında eğitmenlik ve danışmanlık yapmıştır. Bilişim sektöründe Microsoft iş ortağı olan birçok eğitim merkezinde teknik eğitmenlik yapan Çamoğlu, son olarak bir vakıf üniversitesinde Öğretim Görevlisi ve Yazılım Otomasyon Sorumlusu olarak çalışmaktadır.

"SQL Server 2005", "Visual Basic 2008", "Mobil Programlama" ve "Programlama ve Veritabanı Mantığı" başlıklarında yayınlanmış kitapları olan Çamoğlu, Microsoft Yetkili Eğitim Danışmanı (MCLC), Microsoft Yetkili Eğitmeni (MCT), Microsoft Yetkili Yazılım Geliştirme Uzmanı (MCPD) ve Microsoft Yetkili Teknoloji Uzmanıdır (MCTS).

Microsoft tarafından 2007, 2008 ve 2009 yıllarında 3 yıl üst üste "En Değerli Profesyonel" (MVP) unvanına layık görülen yazar ayrıca New Horizons tarafından 2007 yılında "Worldwide Excellence in Training" ödülüne layık bulunmuştur.

EK A: Belge İzleme Etmeni Kodları

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Linq;
using System.ServiceProcess;
using System.Text;

using System.IO;
using ABISCerceve;
using System.Threading;

namespace ABISBelgeIzlemeEtmeni
{
partial class BelgeIzlemeEtmeni : ServiceBase
{
    // izleyiciler listesi için koleksiyon
    List<FileSystemWatcher> izleyiciler
        = new List<FileSystemWatcher>();
    System.Timers.Timer zamanlayici;
    Queue<DegisenDosyaBilgisi> degisenDosyalar
        = new Queue<DegisenDosyaBilgisi>();

    public BelgeIzlemeEtmeni()
    {
        InitializeComponent();
        zamanlayici = new System.Timers.Timer(250);
        zamanlayici.Elapsed
            += new System.Timers.ElapsedEventHandler(zamanlayici_Elapsed);
    }

    void zamanlayici_Elapsed(object sender, System.Timers.ElapsedEventArgs
e)
    {
        if (degisenDosyalar.Count < 1)
        {
            return;
        }

        if (degisenDosyalar.Count > 1)
        {
            // ikili işlem gerçekleştir
            DegisenDosyaBilgisi ddb1 = degisenDosyalar.Dequeue();
            DegisenDosyaBilgisi ddb2 = degisenDosyalar.Dequeue();

            IkiliDegisimBilgisi idb=new IkiliDegisimBilgisi();
            idb.Birinci=ddb1;
            idb.Ikinci=ddb2;

            Thread t = new Thread(new
ParameterizedThreadStart(IkiliDegisimIsle));
            t.Start(idb);
        }
        else if (degisenDosyalar.Count == 1)
```

```
        {
            DegisenDosyaBilgisi ddb = degisenDosyalar.Peek();
            if
(DateTime.Now.Subtract(ddb.DegismeZamani).TotalMilliseconds > 50)
            {
                // tekli işlemi gerçekleştir
                ddb = degisenDosyalar.Dequeue();
                Thread t = new Thread(new
ParameterizedThreadStart(TekliDegisimIsle));
                t.Start(ddb);
            }
        }
    }


    protected override void OnStart(string[] args)
    {
        zamanlayici.Interval = 250;
        zamanlayici.Start();

        // LINQ ile bilgisayardaki sabit disklerin listesi alınıyor
        var diskler = from disk in DriveInfo.GetDrives()
                      where disk.DriveType == DriveType.Fixed
                      select new { disk.Name, disk.VolumeLabel };

        izleyiciler.Clear();

        // her bir disk için izleyici tanımlanarak olaylar bağlanıyor
        foreach (var oge in diskler)
        {
            FileSystemWatcher izleyici = new FileSystemWatcher(oge.Name);
            izleyici.EnableRaisingEvents = true;
            izleyici.NotifyFilter
                = NotifyFilters.FileName | NotifyFilters.DirectoryName;
            izleyici.Filter = "*.pdf";
            izleyici.IncludeSubdirectories = true;
            izleyici.Changed
                += new FileSystemEventHandler(izleyici_Changed);
            izleyici.Created
                += new FileSystemEventHandler(izleyici_Changed);
            izleyici.Deleted
                += new FileSystemEventHandler(izleyici_Changed);
            izleyici.Renamed
                += new RenamedEventHandler(izleyici_Renamed);

            izleyiciler.Add(izleyici);
        }
    }

    void izleyici_Renamed(object sender, RenamedEventArgs e)
    {
        // belgenin veritabanındaki tam adı değiştiriliyor.
        ABISCerceve.DBIslevler.BelgeAdiDegistir(e.OldFullPath, e.FullPath,
e.Name);
    }

    void izleyici_Changed(object sender, FileSystemEventArgs e)
    {
        switch (e.ChangeType)
        {
```

```
            case WatcherChangeTypes.All:
                break;
            case WatcherChangeTypes.Changed:
                break;
            case WatcherChangeTypes.Created:
                DegisenDosyaBilgisi ddb1
                    = new DegisenDosyaBilgisi();
                ddb1.Ad = Path.GetFileName(e.FullPath);
                ddb1.DegisiklikTipi
                    = DegisenDosyaBilgisi.DegismeTipi.Olusturuldu;
                ddb1.TamAd = e.FullPath;
                degisenDosyalar.Enqueue(ddb1);
                break;
            case WatcherChangeTypes.Deleted:
                DegisenDosyaBilgisi ddb2
                    = new DegisenDosyaBilgisi();
                ddb2.Ad = Path.GetFileName(e.FullPath);
                ddb2.DegisiklikTipi
                    = DegisenDosyaBilgisi.DegismeTipi.Silindi;
                ddb2.TamAd = e.FullPath;
                degisenDosyalar.Enqueue(ddb2);
                break;
            case WatcherChangeTypes.Renamed:
                break;
            default:
                break;
        }
    }


    protected override void OnStop()
    {
        zamanlayici.Stop();
        foreach (FileSystemWatcher fsw in izleyiciler)
        {
            fsw.EnableRaisingEvents = false;
            fsw.Dispose();
        }
        izleyiciler.Clear();
        degisenDosyalar.Clear();
    }


    private void IkiliDegisimIsle(object ikiliDegisimBilgisi)
    {
        IkiliDegisimBilgisi degisimBilgisi
            = (IkiliDegisimBilgisi)ikiliDegisimBilgisi;
        DegisenDosyaBilgisi ddb1 = degisimBilgisi.Birinci;
        DegisenDosyaBilgisi ddb2 = degisimBilgisi.Ikinci;

        if
(ddb1.DegisiklikTipi==DegisenDosyaBilgisi.DegismeTipi.Olusturuldu
                        &
ddb2.DegisiklikTipi==DegisenDosyaBilgisi.DegismeTipi.Silindi)
        {   // Kes / Yapıştır (Taşıma)
            // Veritabanından dosya tam adı değiştiriliyor
            ABISCerceve.DBIslevler.BelgeAdiDegistir(ddb2.TamAd, ddb1.TamAd,
ddb1.Ad);
        }
```

```
        if (ddb1.DegisiklikTipi == DegisenDosyaBilgisi.DegismeTipi.Silindi
                        & ddb2.DegisiklikTipi ==
DegisenDosyaBilgisi.DegismeTipi.Olusturuldu)
        {  // Silme ya da geri alma
            if (!ddb1.TamAd.Contains("RECYCLER"))
            {   // Silme işlemi
                // Dosya ve indeks bilgileri veritabanından siliniyor
                ABISCerceve.DBIslevler.BelgeSil(ddb1.TamAd);
            }

            if (!ddb2.TamAd.Contains("RECYCLER"))
            {   // Silinen dosya geri alındı işlemi
                // Dosya, dosyalar listesine ekleniyor
                ABISCerceve.DBIslevler.BelgeEkle(ddb2.TamAd, ddb2.Ad);
            }
        }
    }

    private void TekliDegisimIsle(object degisenDosyaBilgisi)
    {
        // yeni dosya eklendi.
        DegisenDosyaBilgisi degisenDosya =
(DegisenDosyaBilgisi)degisenDosyaBilgisi;
        ABISCerceve.DBIslevler.BelgeEkle(degisenDosya.TamAd,
degisenDosya.Ad);
    }

}

    public class IkiliDegisimBilgisi
    {
        public DegisenDosyaBilgisi Birinci;
        public DegisenDosyaBilgisi Ikinci;
    }

    public class DegisenDosyaBilgisi
    {
        public string Ad;
        public string TamAd;
        public DateTime DegismeZamani;
        public DegismeTipi DegisiklikTipi;

        public enum DegismeTipi
        { Silindi, Olusturuldu, Degisti }
    }


}
```