WEB TABANLI
METİN MÜHENDİSLİĞİ MİMARİSİ
(GOTA)

Mehmet Akif ÇAKAR
Yüksek Lisans Tezi

Bilgisayar Mühendisliği Anabilim Dalı
Haziran 2010

JÜRİ VE ENSTİTÜ ONAYI

Mehmet Akif ÇAKAR'ın "**Web Tabanlı Metin Mühendisliği Mimarisi (GOTA)**" başlıklı **Bilgisayar Mühendisliği** Anabilim Dalındaki, Yüksek Lisans Tezi 28.06.2010 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı-Soyadı	**İmza**
Üye (Tez Danışmanı)	**: Yard. Doç. Dr. ÖZGÜR YILMAZEL**	**.....**
Üye	**: Yard. Doç. Dr. CÜNEYT AKINLAR**	**.....**
Üye	**: Yard. Doç. Dr. KAMİL ÇEKEROL**	**.....**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ÖZET

Yüksek Lisans Tezi

WEB TABANLI METİN MÜHENDİSLİĞİ MİMARİSİ (GOTA)

Mehmet Akif ÇAKAR

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Yard. Doç. Dr. Özgür YILMAZEL
2010, 59 Sayfa

Metin mühendisliği alanında günümüze kadar birçok araştırma yapılmış ve yapılan araştırmalar doğrultusunda uygulamalar geliştirilmiştir. Fakat geliştirilen uygulamaların küçük bir kısmı gelişim evresini sürdürebilmiştir. Bu durumun nedenini sorgulayan çalışmalarda, uygulamaların mimari açıdan eksikliklerinin yanı sıra yazılım kalite ilkelerinin önemi (Clarke, 1996) vurgulanmaktadır.

Bu çalışma metin mühendisliği alanında yeni bir web tabanlı mimari yaklaşım sunmayı amaçlamaktadır. Hedeflenen mimari yeni bir doğal dil işleme modeli önermek yerine, kendini günümüzde kabul görmüş GATE, OpenNLP, LingPipe gibi araçların birikimleri üzerine konumlandırmaktadır. Bu şekilde DDİ alanında sistem mühendisliği, arayüz mühendisliği ve güvenilirlik mühendisliği gibi alanların kazanımları dikkate alınarak yazılım kalite kriterleri açısından da başarılı bir mimari ortaya koyulması hedeflenmektedir. Ortaya koyulan mimariye dayalı olarak geliştirilen sistem üzerinde yapılan testlerle sistemin kararlılığı ve güvenilirliği sınanmış ve kalite testlerinde başarılı sonuçlar elde edilmiştir.

Anahtar Kelimeler: Doğal Dil İşleme, Bilgi Çıkarımı, Metin Mühendisliği, Ölçeklenebilirlik, Kullanıcı arayüzü Mühendisliği, Sistem Mühendisliği

ABSTRACT

Master of Science Thesis

ONLINE TEXT ENGINEERING ARCHITECTURE (GOTA)

Mehmet Akif ÇAKAR

Anadolu University
Graduate School of Sciences
Computer Engineering Program

Supervisor: Yard. Doç. Dr. Özgür YILMAZEL
2010, 59 pages

Many researches have been made in text engineering field until today and tools have been developed in line with these researches. However; very few of the developed tools managed to resume its development stage and provided opportunities that enabled to benefit directly from research outputs. Why software quality criteria's matter is emphasized in a study (Clarke, 1996) that questions the reason of this situation.

This study aims to offer a new architectural approach in text engineering field. The aimed architecture positions itself on build-up of accepted tools such as GATE, OpenNLP, LingPipe rather than offering a new natural language processing model. In this manner it is aimed to introduce an architecture which is also successful in terms of software quality metrics by taking principles of fields such as system engineering in NLP field and one of its sub-disciplines interface engineering, and principles of reliability engineering into consideration. It has been observed that a gain was achieved in the system developed on the basis of introduced architecture in terms of many software quality metrics as the result of measurements performed.

Keywords: Natural Language Processing, Information Retrieval, Text Engineering, Scalability, User Interface Engineering, System Engineering

TEŞEKKÜR

Bu tez çalışmasının başlangıç fikrini ortaya koyan ve yönlendirmeleriyle destek ve katkı sağlayan danışmanım Sayın Yar. Doç. Dr. Özgür YILMAZEL'e, 2210'nolu destek programıyla çalışmaya önemli katkı sağlayan TUBİTAK – Bilim İnsanı Destekleme Daire Başkanlığı'na, akademik süreçteki değerli katkılarından ötürü Sayın Burcu YÜREKLİ'ye ve sıkça sabahlamalara uzanan yoğun çalışma sürecinde verdiği moral&motivasyon desteğinden ötürü eşim Gülten ÇAKAR'a teşekkür ederim.

Mehmet Akif ÇAKAR

Haziran, 2010

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

SİMGELER KISALTMALAR DİZİNİ

BÇ **:** Bilgi Çıkarımı

BNEP **:** Basit Nesne Erişim Protokolü

DDİ **:** Doğal Dil İşleme

GKA **:** Grafiksel Kullanıcı Arayüzü

GOTA **:** Web Tabanlı Metin Mühendisliği Mimarisi'nin kod adı

İDA **:** İçerik Dağıtım Ağı

KSO **:** Kurumsal Servis Otobüsü

WCF **:** Windows Communication Foundation

1. GİRİŞ

İnternet ağı, günden güne hızla genişleyen kullanıcı kitlesi ve bilgi birikimiyle oldukça büyük bir içeriği barındırmaktadır. Web ağı içerisinde bulunan birçok servis ve uygulama aracılığıyla veri toplanmakta ve işlenmektedir. Toplanan verilerin büyük bir çoğunluğunu yapılandırılmamış veriler (Grimes, 2006) oluşturmaktadır. Yapılandırılmamış verilerden oluşan küme içerisinde, istenilen alt veri kümeye ulaşabilmek günümüzde de tam anlamıyla çözülemeyen başlı başına bir sorun olmuştur. Araştırmacılar doğal dil işlemenin alt dallarından faydalanarak, yapılandırılmamış verileri işleyerek yapılandırılmış bilgiye dönüştürmektedirler.

Doğal dil işlemenin alt tekniklerinin başlıcaları bilgiye erişim ve bilgi çıkarımıdır. Bilgiye erişim teknikleri (Grefenstette, 1998; Strzalkowski, 1999; Ricardo A. Baeza-Yates, 1999) genel anlamda, verilen sorguya cevap olarak benzer dokümanları elde etmeye yarar (Jordi, Ageno, & Català, 2006). Uygulama alanı olarak daha çok arama motorlarında kullanılan bilgiye erişim teknikleri bu çalışmada kapsam dışında tutulmuştur.

Bilgi çıkarımı; doğal dilde yazılmış metinlerden ön-tanımlı olay tiplerini, varlık kümelerini veya ilişkileri barındıran alt kümeleri analiz süreci sonunda elde etmeye dayalı bir disiplin olarak tanımlanabilir. Bilgi çıkarım süreci sonunda elde edilen çıktılar anlamsal olarak işaretlenip daha üst model bilgi gösterim platformlarında kullanılabilirler.

Bilgi çıkarımı alanında günümüze kadar geliştirilmiş birçok araç mevcuttur. Fakat bu araçların önemli bir kısmı belirli kapsam dahilinde nokta-atış hedeflere yönelik olarak tasarlanmışlardır. Her ne kadar bu yaklaşım araştırma dünyasında kabul görse de pratik açıdan bakıldığında geliştiren araştırmaların pek az bir kısmı gelişmesine devam edebilmekte ve çıktılarından doğrudan istifade edilebilmektedir. Diğer yandan araçların yazılım kalite kriterlerinde kabul edilmiş ilkeler yerine, yazılımcıların sezgisel yaklaşımlarını kullanarak geliştirilmeleri ölçeklenebilirlik, dağıtık hesaplama, performans optimizasyonu gibi alt disiplin ihtiyaçlarının karşılanmasında zorluk çıkartmaktadır. Geçmiş çalışmalarda gözlemlenen bir başka

sorun ise araçların kullanımı konusunda kullanıcı arayüzü mühendisliği ilkelerinin göz ardı edilmesidir. Bu nedenle geliştirilen araçların kullanımında yeni kullanıcıların zorlandığı söylenebilir. Tespit edilen genel problemlere ek olarak geçmiş çalışmalarda çözülmesi hedeflenen; kaynakların ve sonuçların ortak merkezde toplanması (Eryiğit, 2007), sürümleme ve senkronizasyon gibi birçok alt problemlerde günümüzde tartışılan konular arasında yer almaktadır.

Yapılan araştırmalarda UIMA ve GATE altyapıları DDİ alanında kabul görmüş açık kaynak kodlu projeler olarak tespit edilmiştir. Fakat her iki projenin de mevcut sürümleri yazılım kalite kriterleri açısından arzulanan seviyede değillerdir. Örneğin GATE Geliştirici 5 sürümünün statik metotlarla kodlanmasından dolayı, doğrudan uygulama sunucusu üzerinde yatay ölçeklenmede problemlerle karşılaşılmaktadır. Öte yönden her iki yapının da web tabanlı kurumsal uygulamalarda güvenilir bir biçimde kurulması ve ayarlanması deneyim ve detaylı çalışma gerektiren bir süreçtir. Her iki ortamında güncel sürümlerinde çevrim-içi takım halinde çalışma imkânı yoktur. Fakat GATE Teamware adında rol tabanlı, takım halinde proje geliştirilebilecek bir platform IR-Facility bünyesinde duyurulmuştur. Teamware'ye ek olarak GATE Cloud adında GATE'nin dağıtık hesaplama yapabilecek şekilde paralel programlama ilkeleriyle kodlanan bir sürümü üzerinde çalışıldığı çeşitli yayınlarda tartışılmıştır. Fakat her iki ürün hakkında da henüz çok kısıtlı bilgi olduğundan bu konuda GOTA ile kapsamlı karşılaştırma yapılamamıştır.

GOTA, yukarıda bahsedilen problemlere çözüm önerme noktasında başlatılmış bir çalışmadır. Çıkış noktası doğrultusunda iki temel hedef belirlenmiştir. İlk hedef; araştırma dünyasındaki kazanımlardan faydalanabilmek için mevcut DDİ altyapıları üzerinde çalışabilme imkanının sunulmasıdır. DDİ altyapıları üzerinde çalışırken aynı anda birden çok araç üzerinde hibrit proje geliştirilebilmesi hedeflenmiştir. İkinci hedef ise sektörel deneyimlerde elde edilen pratik kazanımları akademik kazanımlarla birleştirerek daha sağlam, ölçeklenebilir ve güvenilebilir bir platform oluşturmaktır. GOTA projesinde istenilen hedeflere ulaşabilmek için sektörel ve bilimsel alanlarda elde edilen kazanımların birlikte değerlendirilmesi gerektiğine inanılmaktadır. Bu noktadaki motivasyon kaynağını geçmiş araştırmalar

üzerinde yapılan ve yazılım kalitesinin önemini gösteren çalışmayla (Clarke, 1996) özetlenebilir.

Bu tez çalışmasında, GOTA kod adlı mimariye dayalı olarak bilgi çıkarım platformu geliştirilmiş ve elde edilen kazanımlar ortaya konulmuştur. İlerleyen bölümlerde sırasıyla mevcut bilgi çıkarım platformları incelenecek ardından GOTA platformunun mimarisi, geliştirilme ilkeleri, karşılaşılan problemler, getirilen çözüm önerileri, yapılan testler, bilgi edinimleri ve süreç çıktılarına değinilecektir. Son bölümde sonuç ve önerilere değinilerek çalışma sonlandırılacaktır.

2. BİLGİ ÇIKARIM PLATFORMLARI

Günümüzde araştırmacıların ve yazılım geliştiricilerinin BÇ uygulamalarını modellemek ve geliştirmek için kullanabilecekleri birçok platform ve altyapı mevcuttur. En basit anlamda küçük masaüstü araçlardan oluşan BÇ platformları, günümüzde paralel bilgisayarlar için tasarlanmış dağıtık yazılım sistemlerine dek uzanmıştır. Dağıtık sistemlere ek olarak teknolojinin gelişmesiyle birlikte mevcut platformlardan beklentiler artmakta ve artan beklentilerin etkisi sonucunda yeni çalışma ortamlarının ortaya çıkması sağlanmaktadır.

BÇ platformları birtakım niteliklere göre alt başlıklar altında incelenebilir. Bu gruplandırmalardan başlıcaları çalışma ortamlarıdır. BÇ platformları çalışma ortamlarına göre Bölüm 2.1'de incelenmiştir. GOTA projesinde BÇ ve DDİ altyapısı olarak kullanılan GATE altyapısı Bölüm 2.2'de ele alınmıştır.

2.1. Çalışma Ortamlarına Göre BÇ Platformları

Çalışma ortamlarına göre BÇ platformları iki başlık altında ele alınmaktadır. Masaüstü platformlar daha çok bireysel araştırmalarda kullanılırken, web tabanlı platformlar birden çok kullanıcının erişimine imkân verdiğinden takım halinde proje geliştirmekte kullanılırlar. Her iki platformda detaylı olarak incelenirse;

2.1.1. Masaüstü platformlar

Günümüzde varlığını sürdüren platformların büyük çoğunluğu masaüstü platformlar altında incelenebilir. Masaüstü platformların genel özelliği aynı anda tek kullanıcıyla etkileşime geçebilecek şekilde tasarlanmalarıdır. Masaüstü platformların bu şekilde tasarlanmasındaki etkenlerin biride akademik kapsamda gerçekleştirilen araştırmalar için yeterli imkânı sunmalarıdır. Yapılan araştırmalarda masaüstü platformların geliştirilmesinde statik değişkenlerin ve durum bağımlı metotların kullanımına sıkça rastlanmıştır. Bu nedenle eşzamanlılık, ölçeklenebilirlik gibi kavramların göz ardı edildiği söylenebilir. Masaüstü platformlarda eşzamanlılığın göz ardı edilmesinden dolayı takım halinde aynı uygulama üzerinde asenkron çalışabilmek imkânsız bir hal almıştır. Bu soruna çözüm olarak istemci-sunucu

mimarisine dayanan masaüstü platformların geliştirilmesi teknik açıdan mümkün olmasına rağmen bu yönde bir çalışmaya yapılan araştırmalarda rastlanılamamıştır.

2.1.2. Web tabanlı platformlar

Takım halinde eşzamanlı çalışabilme ihtiyacının yanı sıra ortak erişim arayüzü, kaynakların tek merkezde toplanması gibi birçok etken web tabanlı DDİ platformlarının ortaya çıkmasını sağlamıştır. Günümüzde az sayıda web tabanlı platform bulunmasına rağmen bu yönde bir eğilim olduğu gözlemlenmiştir. Mevcut web tabanlı platformlar üzerinde yapılan araştırmada yalnızca GATE Teamware adlı ürünün doğrudan takım halinde proje geliştirmeye imkân sağladığı tespit edilmiştir. Fakat mevcut platformlar hakkında çok kısıtlı bilgi kaynağı mevcut olduğundan derinlemesine bir inceleme çalışması yapılamamıştır.

GOTA, web tabanlı platform başlığı altında ele alınabilecek bir çalışmadır. Dolayısıyla platform üzerinde takım halinde proje geliştirmeye imkân sağlanmasının yanı sıra ortak ve standart bir erişim arayüzünün sunulması gibi birçok gereksinim karşılanmaktadır.

2.2. GATE - Metin Mühendisliği İçin Genel Mimari

GATE (Cunningham, Maynard, Bontcheva, & Tablan, GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications., 2002), Sheffield Üniversitesi tarafından doğal dil işleme alanı için geliştirilmiş bir platformdur. Araştırma altyapısı Cunningham'ın doktora tezine (Cunningham, Software Architecture for Language Engineering, 2000) dayanmaktadır. GATE, 1996 yılında yayımlanmasından itibaren birçok BÇ ve DDİ uygulamasında kullanılmıştır (Maynard, et al., 2000).

GATE, DDİ araştırmacılarına üç temel öğeden oluşan yazılım arayüzü sunar. Bu üç öğe:

- doğal dil işleme sistemi tasarlarken bileşenlerin ne şekilde geliştirilmesi gerektiğini tanımlayan bir *mimari,*

- tanımlanan mimariye dayanan yazılım geliştiricileri için java dili kullanılarak kodlanmış, temel işlemleri barındıran bir *altyapı,*
- doğal dil araştırmacıları için birçok bileşen ve aracı kapsayan, tanımlı altyapının üzerine geliştirilmiş grafiksel *geliştirme ortamı*dır.

GATE çeşitli DDİ metodolojileri ve ontolojik kaynaklara tekbiçimlilik ilkesi çerçevesinde ulaşabilmeyi hedefler. Dolayısıyla ANNIE, OpenNLP gibi birçok farklı topluluk tarafından geliştirilmiş ortamlara GATE üzerinden benzer yöntem kullanılarak ulaşılabilinmektedir. Yeni geliştirilen kaynaklara ek olarak önceden geliştirilen birçok kaynakta tanımlı ilkeler çerçevesinde, nesneye yönelik yazılım geliştirme yaklaşımına göre tekrar modellenip sadeleştirilerek dil mühendislerinin kullanımına sunulmuştur. Günlük araştırma döngüsü içerisindeki depolama, veri görselleştirme, farklı işleme kaynaklarının yüklenmesi gibi veri üzerinde gerçekleştirilen süreçler GATE kullanan dil araştırmacıları için şeffaflaştırılmış, doğrudan müdahale edilebilir hale getirilmiştir.

2.2.1. GATE mimarisinde kaynaklar

GATE mimarisinde kaynaklar üç başlık altında incelenir bunlar:

- *Dil kaynakları*: sözlükler, ham veriler, dokümanlar, ontolojiler gibi kaynakları barındırır.
- *İşleme kaynakları*: dil kaynaklarını işleyen işaretleyici, ayrıştırıcı gibi bileşenlerdir.
- *Görsel kaynaklar*: işleme ve dil kaynakları üzerinde işlem yapabilmek için görsel kullanıcı arayüzü bileşenleridir.

Bu üç tür kaynak GATE sistemi üzerinde dil işleme çalışması yapılırken kullanılan bileşen modelidir. Dil işleme kaynakları genel olarak CREOLE(ing. the Collection of REusable Objects for Language Engineering) olarak adlandırılır. Dil mühendisleri yeni işleme kaynaklarını geliştirip GATE mimarisine tak-çıkar mantığı içerisinde entegre edebilir. GATE kullanıcı klavuzunda (Cunningham, et al.,

Developing Language Processing Components with GATE, 2009) detaylı olarak incelenen bu süreç araştırmalarda sıkça gereksinim duyulan bir ihtiyaçtır.

2.2.2. Uygulamalar ve veri ambarları

Uygulamalar ve veri ambarları GATE tarafından DDİ araştırmacılarına ek olarak sunulan iki ayrı yapıdır. İşleme kaynaklarının gruplandırılarak set halinde dil kaynakları üzerinde çalıştırılması, GATE terminolojisinde uygulama olarak adlandırılmaktadır. Uygulamalar işleyeceği dil kaynağının ve çalışmanın türüne göre de dallara ayrılır. Dil kaynağı türüne göre genel ve toplu uygulama kanalları olarak iki başlık altında incelenebilir. Genel kanallar herhangi türden dil kaynaklarını işlemek için kullanılırken toplu kanallar benzer türden dil kaynaklarını toplu olarak işlemek için kullanılır.

Veri ambarları ise araştırmalar da devamlı olarak kullanılan dil kaynaklarının kalıcı bir biçimde saklanmasını sağlar. Proje geliştirilirken yapılan değişikliklerin çıktılarını karşılaştırabilmek için uygulamanın aynı dil kaynağı üzerinde çalıştırılması çok yaygın bir ihtiyaç olduğundan GATE mimarisinde veri ambarları tasarlanmıştır. Günümüzde GATE iki tür veri ambarını desteklemektedir; Seri Veri Ambarı(ing. Serial Datastore) dil kaynaklarını direk olarak dosya sisteminde saklarken Veritabanı Veri Ambarı(ing. Database Datastore) kaynakları saklamak için ilişkisel veritabanı yönetim sistemlerini kullanmaktadır. Her iki veri ambarında da dil kaynaklarına erişim gerektiğinde kısmi ön bellekleme yapılabilmektedir. Bu sayede uygulamalar çalıştırıldığında dil kaynakları kısmen bellekten okunduğu için işlem süresi kısaltılmış olmaktadır.

2.2.3. Ek açıklamalar

Dil kaynakları üzerinde çalışan işleme kaynakları, uygulama sürecinin ardından orijinal metin hakkında birtakım bilgiler üretir. Örneğin; simge işlemcisi(ing. tagger) kelime, sayı, noktala işareti gibi bilgiler üretirken, konuşma parçası işlemcisi(ing. POS tagger) kelime, isim, zamir, fiil gibi bilgiler üretir. Üretilen bu bilgiler ek açıklamalar(ing. annotations) olarak adlandırılır.

Ek açıklamalar, gömülü ve referans açıklama olarak iki türlü saklanır (Cunningham, Software Architecture for Language Engineering, 2000). Gömülü saklama yönteminde işlem çıktıları SGML, XML veya diğer biçimleme dilleriyle(ing. markup language) işaretlenerek orijinal metin içerisinde ek açıklama değerleriyle birlikte saklanır. Referans saklama yönteminde ise orijinal metin ve ek açıklama değerleri farklı dosyalarda tutulur Referans saklama yöntemi GATE tarafından kabullenilen bir çözümdür.

GATE ek açıklamalarında dört temel bilgi bulunur:

- *ek açıklamanın türü* farklı DDİ yapılarının kullanıma göre değişebilen bir bilgidir. Örneğin ANNIE ve OpenNLP tarafından üretilen ek açıklamalarda, kullanılan işleme kaynakları açıklama türü alanına değişik bilgiler kaydedebilmektedir.
- dokümanda tekil olan ve yalnızca ilgili ek açıklamaya ait olan *ID* değeri.
- ek açıklama değerinin *başlangıç ve bitiş noktasını gösteren* işaret değerleri
- ek açıklamanın niteliklerini barındıran ad ve değerlerden oluşan *ek bilgiler listesi*

Başta GATE olmak üzere mevcut BÇ platformları GOTA platformunda işlem sunucusu olarak konumlandırılabilmektedirler. Bölüm 2.2'de masaüstü platform olarak incelenen GATE platformu, web tabanlı platform olan GOTA'nın mevcut sürümüne başarıyla entegre edilmiş ve üzerinde örnek BÇ uygulamaları geliştirilmiştir. Entegrasyon süreci ve GOTA mimarisi Bölüm 3'te detaylı olarak ele alınmıştır.

3. GOTA PLATFORMU

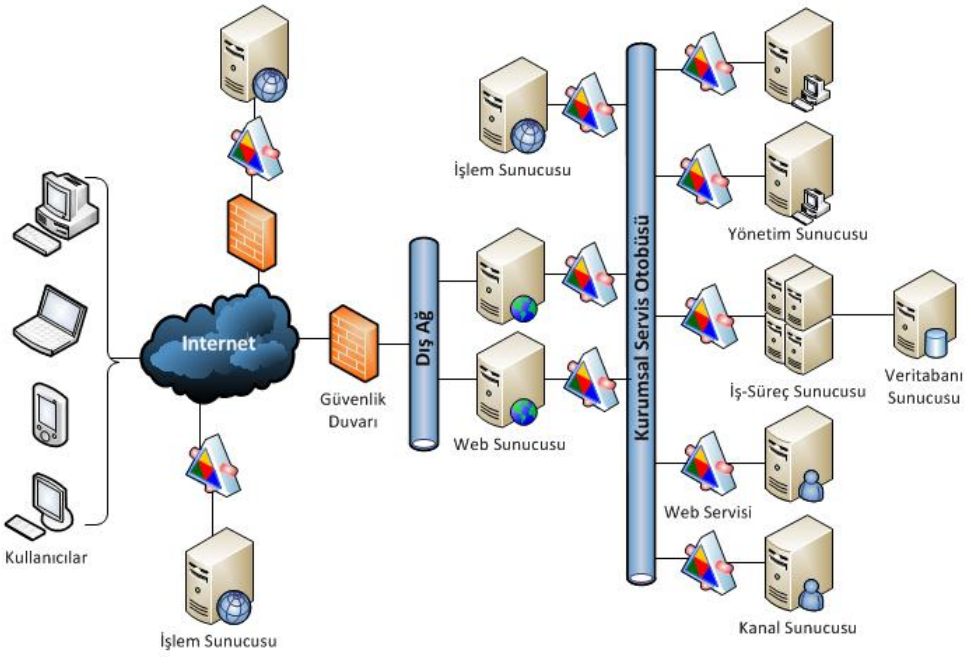
GOTA, BÇ projelerinin geliştirilmesini doğrudan desteklemekle birlikte diğer DDİ alanlarında da projelerin geliştirilmesine imkân verebilecek şekilde tasarlanan genel bir mimaridir. Pratik anlamda ise GOTA; tasarlanan mimariye dayalı web ortamına uygun olarak geliştirilen platformu temsil etmektedir. Geliştirilen platformda, platform bağımsızlığı ve servis tabanlı mimari olmak üzere iki temel ilke gözetilmiştir. Modelleme ve geliştirme sürecinde karşılaşılan problemlerin çözümünde bu temel ilkeler gözetilerek yol alınmıştır.

3.1. Mimari Tasarım

GOTA platformu, servis tabanlı mimari prensiplerine uygun olarak dağıtık bileşen mimarisiyle tasarlanmıştır. Sistemde her bir görev gereksinimi belirlenerek ayrı ayrı alt bileşenler tasarlanmıştır. Bu yönteme dayanarak tasarlanan prototipin mevcut sürümünde beş ayrı bileşen türü elde edilmiştir. Bunlar sırasıyla yönetim sunucusu, web sunucu, iş-süreç sunucusu, kanal sunucusu ve işlem sunucusudur. Geliştirilen bileşenler dışında hazır ürün olarak kullanılan kurumsal servis otobüsü (Chappell D. , Enterprise Service Bus, 2004)(ing. enterprise service bus), veritabanı sunucusu gibi harici bileşenlerden faydalanılmıştır.

Bileşenlerin kendi aralarındaki iletişim, web servisler aracılığıyla KSO üzerinden sağlamaktadır. İletişimi web servisler aracılığıyla gerçekleştirmekteki temel amaç geliştirilen platformun erişilebilirliğini, geliştirme ortamının değişkenlerinden soyutlayarak ileride entegre olacak uygulamalara daha geniş imkânlar sağlamaktır. İletişimde KSO kullanılmasındaki temel gerekçe ise KSO'lerinin sunduğu zengin adaptör çeşitliliğiyle ileride ortaya çıkabilecek entegrasyon ihtiyaçlarına hızlıca cevap verebilme imkanı sunması ve KSO'lerinin temel karakteristiklerinden olan geri izleme, kayıt tutma gibi yönetimsel fonksiyonlara ek olarak; sunulan hareket yönetimi, güvenilir iletişim, güvenlik gibi birçok kalite servislerinin mevcudiyetidir. Mevcut prototipte kullanılan KSO kümelemeye dayalı olarak kurulmadığı için KSO'nün arızalanması halinde sistemin durması (Dooley,

2002)(ing. single point of failure) dezavantajını barındırmaktadır. Tanımlanan ağ mimarisi Şekil 3.1'de gösterilmiştir.



Şekil 3.1 - GOTA Ağ Mimarisi

Sistemin bileşenlerini sırayla incelemek gerekirse;

- Yönetim Sunucusu: Sistemdeki parametreleri ve süreçleri ayarlayıp modelleyebilmek için gerekli yönetimsel araçları barındıran bileşendir. Sistemde doğrulama ve yetkilendirme gibi fonksiyonları barındıran servislerde bu bileşenin içerisinde yer alır. Yönetimsel işlemlere ek olarak sistemin kararlılık durumunu tespit etmek için, diğer bileşenler üzerinde sağlık kontrolünü (ing. health check) gerçekleştirir. Sağlık kontrolü sırasında sorun görülen bileşenleri pasif olarak işaretler, bu şekilde gelen taleplerin sağlıklı bileşenlere yönlendirilmesini sağlayarak sistemin kararlılığını korur. Yönetim sunucusuna erişim yalnızca WCF web servisleri aracılığıyla gerçekleşmektedir. Bu nedenle sistem yöneticileri bu bileşen üzerinde bir işlem yapmak istediklerinde web sunucusunun grafiksel kullanıcı arayüzünü kullanmaları gerekmektedir.

- Web Sunucusu: Araştırmacılara ve son kullanıcılara web tarayıcıları ve geliştirme ortamları aracılığıyla erişebilecekleri GKA ve BNEP web servislerini sunan bileşendir. Sistemin dış kullanıcılara açılan kapısı olan web sunucusu aynı zamanda KSO ağına yönetimsel ihtiyaçlarda kullanılmak üzere web servisleri sunar. Web sunucusu kullanıcı arabirimine ait fonksiyonlar dışında herhangi bir hesaplama veya mantıksal süreç barındırmamaktadır.
- İş-Süreç Sunucusu: Sistemde KSO ile birlikte omurga görevini gören bileşendir. Tüm bileşen türlerinde kullanılabilecek genel hesaplamalar ve fonksiyonlar iş-süreç sunucusu içerisinde bulunur. Veritabanına doğrudan erişim hakkı yalnız iş-süreç sunucusunda olduğu için diğer bileşenlerin ihtiyaç duyduğu veritabanı kullanımı bileşen üzerindeki web servisleri aracılığıyla gerçekleştirilebilmektedir.
- Kanal Sunucusu: DDİ taleplerinin GKA dışında farklı yöntemlerle işlenebilmesi için geliştirilmiş bileşen türüdür. GOTA'nın mevcut prototipinde BNEP servis kanal türüne ek olarak, ftp, e-posta, messenger gibi dört farklı iletişim arayüzü sunmaktadır. Her bir kanal sunucusu sunduğu iletişim türüne göre farklı yapılar barındırmaktadır. Örneğin; ftp kanal sunucusu yüklü miktarda veriyi işlemek için seri veri ambarı barındırırken, e-posta kanalı veriyi doğrudan bellek üzerinde işlemeye uygun bileşenleri barındırmaktadır.
- İşlem Sunucusu: Doğal dil işleme süreçlerinin gerçekleştirildiği bileşendir. İşlem sunucuları farklı platformlarda geliştirilmiş DDİ altyapılarını barındırabilirler. Örneğin; mevcut sürümde GATE altyapısı Glassfish uygulama sunucusu üzerinde yayımlanarak web servis adaptörü aracılığıyla KSO'ne entegre edilmiştir. Bu şekilde oluşturulan bileşenler aynı zamanda BÇ sunucusu olarakta adlandırılmıştır. Araştırmalarda kullanılan diğer DDİ araçları GOTA platformuna eklenmek istenildiğinde doğrudan web servislerle veya mevcut bir KSO adaptörüyle işlem sunucusu türünde entegre edilebilirler. İşlem sunucularını diğer bileşenlerden ayıran başlıca özellik ise ağ mimarisinde gerek KSO ağında gerekse de internet üzerinde herhangi bir noktada çalıştırılabilmeleridir.

Bölüm 3.2'de detaylı olarak ele alınan ilkeler çerçevesinde tasarlanan GOTA mimarisinin dağıtık bileşen yapısı platformu DDİ işleme teknolojilerinden bağımsız kılmaktadır. Bu şekilde kullanılan DDİ altyapısı tamamen değiştirilebileceği gibi farklı araçlardan da faydalanılabilinmektedir. Örneğin; mevcut mimari tasarımına gelecekte ses-yazı çevrimini gerçekleştiren bir araç entegre edilebilir ve bu araç kullanılarak ses verisi üzerinden BÇ projeleri gerçekleştirilebilir. Örneklemde anlatılan entegrasyon sürecinin diğer bileşenlerin işleyişinde herhangi bir aksamaya neden olmayacağı birçok defa testlerle doğrulanmıştır.

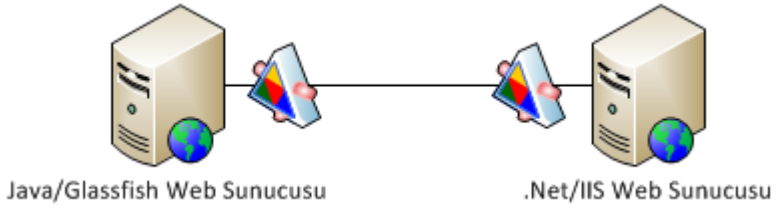
3.2. İlkeler

GOTA mimarisinin tasarımında iki temel ilke gözetilmiştir. Bunlar platform bağımsızlığı ve servis tabanlı mimarinin kullanımıdır. Her iki ilke detaylı olarak ele alındığında;

3.2.1. Platform bağımsızlığı

Platform bağımsızlığı farklı disiplinlerde farklı anlamlarda kullanılabilen bir kavramdır. GOTA mimarisinde platform bağımsızlığı, kullanıcı türüne göre iki açıdan ele alınmaktadır. Bunlar;

- Yazılım geliştiricileri açısından: GOTA projesi temelde .Net ve Java teknolojileri kullanılarak geliştirilmiştir. İki farklı platform arasındaki iletişim üzerine yapılan bir çalışmada (Fisher, Lai, Sharma, & Moroney, 2006) önerilen yöntem olan web servislerinin kullanımı geliştirilen mimaride de kabul edilmiş bir yaklaşım olarak değerlendirilmiştir. Şekil 3.2'de gösterilen bileşenler arası iletişimde, bileşenler arası iletişim web sunucularında barındırılan web servisleri aracılığıyla gerçekleşmektedir.



Şekil 3.2 GOTA bileşenleri arasındaki iletişim

Bu şekilde iki farklı yazılım geliştirme platformu olan java ve .net'e ek olarak BNEP (W3C, 2009) standardını destekleyen diğer tüm yazılım platformları da; GOTA üzerinde geliştirilen DDİ projelerine web servisleri üzerinden erişebilmektedirler.

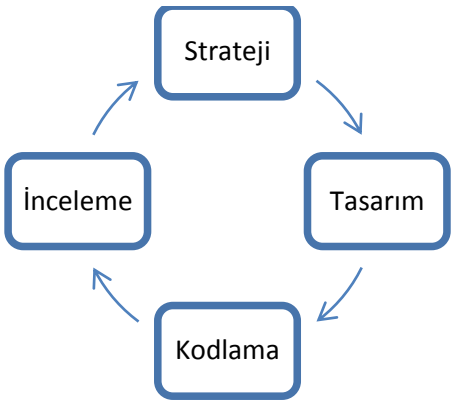
- Araştırmacılar açısından: Araştırmacılar GOTA platformunda aynı zamanda son kullanıcılar olarakta adlandırılmaktadır. Son kullanıcılar açısından platform bağımsızlığı; kullanılan işletim sisteminden ve DDİ platformundan bağımsız olarak uygulamaya erişebilmeyi temsil etmektedir. Bölüm 3.3.4'te detaylı olarak incelenen Ext JS teknolojisinin, son kullanıcıların platform ve tarayıcı bağımsız olarak sisteme erişebilmelerine imkân sağladığı tespit edilmiştir. DDİ platformlarından bağımsızlık ise araştırmacıların dilediklerinde kullandıkları üçüncül araçları GOTA platformuna entegre edebilmelerine imkân sağlanmasıdır. GOTA'nın mevcut sürümünde entegrasyon süreci yazılım geliştiricilerin desteğiyle sağlanabilmektedir.

3.2.2. Servis tabanlı mimari

GOTA platformunda birçok farklı fonksiyonaliteyi barındıran bileşenlerin geliştirilmesine ihtiyaç duyulmaktadır. Gelecekte de birçok platformun sisteme entegre edilebilmesi muhtemel olduğu göz önüne alındığında bileşenlerin birbirine doğrudan bağımlı olarak tasarlanmasının ciddi sorunlara yol açacağı görülmektedir. Bu nedenle bileşenler arası gevşek bağın kurulabilmesi ve ihtiyaç duyulduğunda da yeniden kullanılabilirlik ilkesi çerçevesinde karşılıklı iletişimin sağlanabilmesi gerekmektedir. Bileşenlerin yeniden kullanılabilirliğinin sunduğu avantajlar bazı çalışmalara (Webservices.org, 2005; Keith, 2005) konu olmuştur. Servis tabanlı mimaride yeniden kullanılabilirliğin başarıyla gerçekleştirilebilmesi sürecinde

karşılaşılan sorunlar da birçok çalışmada (Chappell D. , 2006; Fricko, 2006; McKendrick, 2006) tartışılmıştır. Tüm bu çalışmalardan hareketle GOTA projesinde de bileşenlerin karşılıklı kullanım ihtiyacı göz önüne alındığında servis tabanlı mimarinin seçilmesi uygun bulunmuştur. Aynı zamanda bileşenler arası gevşek bağ ilkesi sayesinde karşılıklı iletişim şemasındaki değişiklikler hariç bileşenlerin bağımsız olarak geliştirilebilmesi mümkündür. GOTA'nın mevcut sürümünde birçok bileşen farklı hızlarda geliştirilmesine rağmen kurulum süreçlerinde uygulamanın aralıksız çalışması sağlanabilmektedir.

Şekil 3.3'de gösterilen servis tabanlı mimarinin sunduğu yaşam döngüsü bir yandan kesintisiz gelişim evresini sürdürürken diğer yandan sistemde eksikliği hissedilen ihtiyaçların tespiti için süreçler sunar bu süreçler incelendiğinde;



Şekil 3.3 - GOTA projesinde servis tabanlı mimari yaşam döngüsü

- Strateji: Bileşenlerin tasarım evresinden önce sistemsel mimari yaklaşımın ve bileşenler arasındaki iletişimin ve bağımlılık düzeyinin belirlendiği evredir. Bu evrede sistemdeki yeni ihtiyaçların belirlenmesinin yanı sıra yeni oluşacak durumda atıl kalan veya kalabilecek bileşenler içinde uygun yaklaşımlar belirlenir. Bileşenler üzerinde herhangi bir modelleme veya tasarım çalışması bu evrede yer almaz.
- Tasarım: Strateji evresinde belirlenen hedefler ve yaklaşımlar doğrultusunda bileşenlerin tasarlanması, ihtiyaca uygun iletişim teknolojisinin belirlenmesi gibi mimari aktiviteler bu evrede gerçekleştirilir.

- Kodlama: Tasarım evresinde belirlenen mimari çerçevesinde kodların yazılması evresidir. Geliştirilen kodlar üzerinde birim testleri uygulandıktan sonra çalışan sistemde kesintiye meydan vermeden kurulum yapılması da kodlama evresinin kapsamı dahilinde ele alınır.
- İnceleme: Sistemin çalışması esnasında elde edilen verilerin değerlendirilmesi olarak özetlenebilir. Genelde performans sayaçları, hata kayıt kütüphaneleri, istatistik kayıtları aracılığıyla toplanan veriler strateji evresine kaynak oluştururlar.

Projede dört temel evrede ele alınan servis tabanlı yaşam döngüsünün uygulanmasıyla, geliştirme sürecinde teknik açıdan birçok kazanım sağlanmıştır. Bu kazanımlar sayesinde sistemde yalnızca geliştirilen bileşen üzerinde odaklanılabilmiş, bileşenler arası zayıf bağ sayesinde geliştirilen modülün diğer bileşenlerin çalışmasına doğrudan olumsuz etkisinin önüne geçilmiştir.

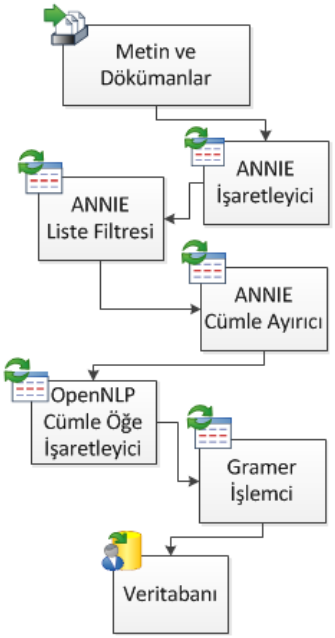
3.3. Problemler ve Çözümler

GOTA platformunun geliştirilmesi esnasında gerek geçmiş çalışmalarda eksikliği hissedilen ihtiyaçların karşılanmasında gerekse teknolojik yeniliklere uyum sağlanması esnasında birçok problemle karşılaşılmıştır. Karşılaşılan problemler ve problemlerin çözümünde elde edilen kazanımlar detaylı olarak incelenirse;

3.3.1. Hibrit platform desteği

Doğal dil işleme süreçlerinin geliştirilebilmesi için ANNIE, OpenNLP, LingPipe (Alias-i, 2008) gibi birçok araç geliştirilmiştir. Her bir aracın kendi alanında, güçlü ve zayıf olduğu yönleri bulunmaktadır. Bu noktadan hareketle farklı platformların aynı BÇ projesinde kullanımının daha başarılı DDİ projelerinin geliştirilebilmesin imkân vereceği sonucuna varılabilir. Fakat farklı araçların aynı BÇ sürecinde kullanılabilmesi karmaşık çözüm gerektiren bir problemdir. GOTA platformunda farklı araçları aynı BÇ projesinde kullanabilmek için ortak girdi ve çıktı şeması belirlenmiştir. Bu yöntemle her bir DDİ aracına özel olarak yazılan modüller, girdi verileri ve işlem çıktıları üzeride dönüştürme işlemi uygulayarak sürecin güvenilirliğini ve kararlılığını

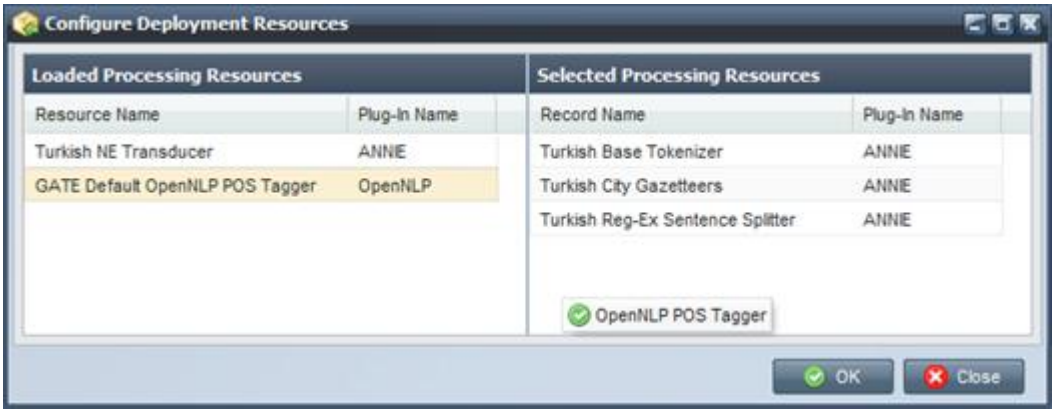
sağlar. Örneğin Şekil 3.4'de ANNIE ve OpenNLP kullanılarak geliştirilen hibrit bir projenin akış diyagramı görülmektedir.



Şekil 3.4 - GOTA üzerinde geliştirilen hibrit proje akış diyagramı

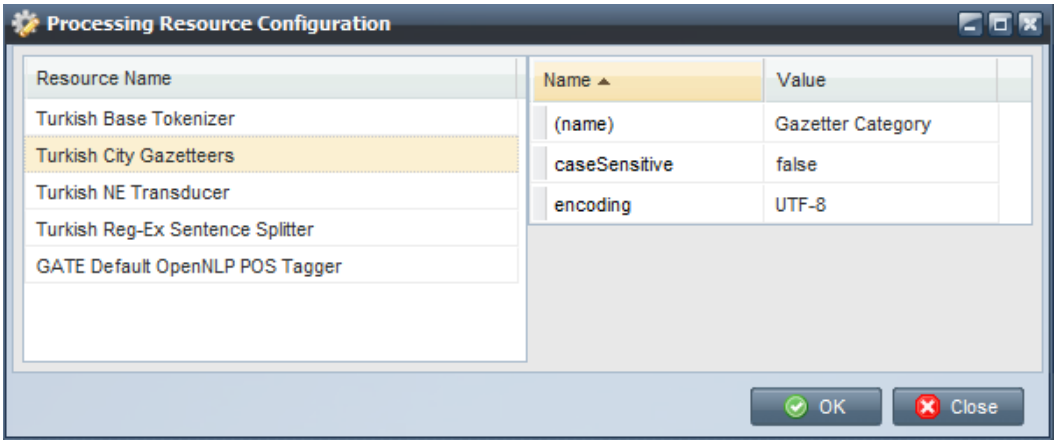
Girdi olarak metin ve dokümanları alarak başlayan işleme süreci; sırasıyla ANNIE işaretleyicisi, ANNIE liste filtresi, ANNIE cümle ayırıcı, OpenNLP cümle öğe işaretleyicisi ve JAPE gramer işlemcisinden geçerek elde edilen çıktıları veritabanına kaydederek sonlanır.

Yukarıda örneklemde anlatılan süreçte her bir bileşeninin işlem sıralaması değiştirilebilmektedir. Şekil 3.5'te görülen hibrit bileşen süreç düzenleme ekranı ile farklı platformlara ait alt bileşenler, sürükle bırak yöntemiyle eklenebilir veya var olan bileşenler işlem listesinden çıkartılabilir.



Şekil 3.5 - Hibrit bileşen süreç düzenleme ekranı

İşleme kaynaklarının yalnızca işleme sürecinde sırasının ve kullanım şeklinin belirtilmesi yetersiz kalmaktadır. Bunun nedeni işleme kaynaklarına ait birden çok niteliğin bulunabilmesidir. Kodlama formatı, karakter duyarlılığı gibi birçok niteliğin projelerin geliştirilmesinden önce düzenlenmesi gereğine binaen Şekil 3.6'de görülen işleme kaynağı nitelik düzenleme ekranı geliştirilmiştir.



Şekil 3.6 - İşleme kaynağı nitelik düzenleme ekranı

DDİ araçlarının aynı proje içerisinde hibrit kullanımına getirilen çözüm ile GOTA'nın daha başarılı BÇ projelerinin geliştirilmesine imkân sağladığı gözlemlenmiştir.

3.3.2. Yoğun işleme

GOTA platformunda geliştirilen metin mühendisliği projelerinde oluşan her bir atomik hareket *işlem* olarak adlandırılmaktadır. Birden çok kullanıcının eşzamanlı çalışması sırasında yoğun işlem yükü oluşabilmektedir. Bölüm 3.4.2'de yapılan stres

testleri yoğun miktarda işlem yükünün sistemi kararsızlaştırdığını göstermiştir. Dolayısıyla çözülmesi mutlak önem arz eden olası aşırı yüklenme durumlarında gelen yükün dağıtılması gerekmektedir. Dağıtım yapılırken de önceden belirlenen birtakım parametrelerin kullanılması ve parametrelerden elde edilen sonuca göre gelen yükün işlem sunucularına dağıtılması gerekmektedir. Yoğun işlem yükünün dağıtılmasına çözüm olarak bir dağıtım algoritması geliştirilmiştir. Geliştirilen dağıtım algoritması, yatay eksende dağıtım gerçekleştirmeye ek olarak dikey eksende dağıtım yapabilmektedir. Dikey dağıtımın gerçekleştirilebilmesi için yük dağıtım sunucusunun içerisinde ve işlem sunucusu üzerinde barındırılan araçlar birlikte kullanılmıştır. Elde edilen algoritma yük dağıtımını öncelikle dikey eksende sürdürmektedir, hedef sunucudaki yük eşik değere ulaştığında ise yatay eksende diğer işlem sunucusuna geçilmektedir. Eğer sistemin taşıyabileceğinden fazla miktarda yük meydana gelirse fazla gelen talepler reddedilmektedir. Bu şekilde sistemin kararlılığı sağlanmış olmaktadır.

Yoğun işleme başlığı altında cevaplanması gereken bir diğer problemde birden çok dil kaynağının toplu olarak işlem kaynakları üzerinde uygulanmasının nasıl sağlanacağıdır. Bu soruna çözüm olarak GOTA sisteminde toplu işleme birimleri tasarlanmıştır. Mevcut sürümde BNEP web servisleri kullanılarak gerçekleştirilebilen toplu işleme süreçleri, verileri ön-bellekleme yöntemiyle toplu olarak işleyebilmektedir.

3.3.3. Kaynakların senkronizasyonu

GOTA platformu takım halinde metin mühendisliği projelerinin geliştirilmesine imkân verdiğinden, aynı kaynak üzerinde birden çok kişinin eşzamanlı çalışabilmesi de mümkündür. Eşzamanlı çalışma sırasında kullanıcı bazlı yapılan değişikliklerin, projenin tümünde anında geçerli olup olmayacağı sorusu sistemin modellemesinde çözülmesi gereken problemler arasında yer almıştır.

Daha çok çoklu-medya uygulamalarında baş gösteren bir problem olan kaynakların senkronizasyonu için birtakım protokoller (Baqai, 1998) tasarlanmıştır. Araştırma ortamı gibi gerçek-zamanlı dağıtık hesaplamaların yapıldığı yapılarda kaynakların senkronizasyonu ise kendi başına araştırma konusudur (Zhang & Cordes,

2004). İncelenen çalışmalarda, olası kaynak çakışmalarının giderilmesi cevaplanamayan veya detaylıca değinilmeyen bir konu olarak kalmıştır. Araştırma ortamında önceden öngörülmesi mümkün olmayan birçok kaynak türünün var olması; çakışma, birleştirme, ayrıştırma gibi olası senkronizasyon problemlerinin genel bir çözümle giderilmesini zorlaştırmaktadır. Yapılan önermelerde ve gerçekleştirilen testlerde özellikle ikili-dosya formatında saklanan dil kaynakları üzerinde çakışma gerçekleştiğinde, son değişiklik yapan kullanıcıya ait kaynağın öncelikli olarak değerlendirilmesi gibi bir çözüm denenmiş fakat bu çözüm de her halükârda ilk kullanıcının ürettiği değerlerin kaybolmasına yol açmıştır. İkili-veri biçiminin kullanıcı müdahalesine doğrudan açık olmaması göz önüne alındığında el müdahalesi yoluyla çözüm yolu da kapanmıştır. Bu ve bunun gibi birçok sorun kaynak senkronizasyonu probleminin çözümünü zorlaştıran etkenler arasında değerlendirilmiştir.

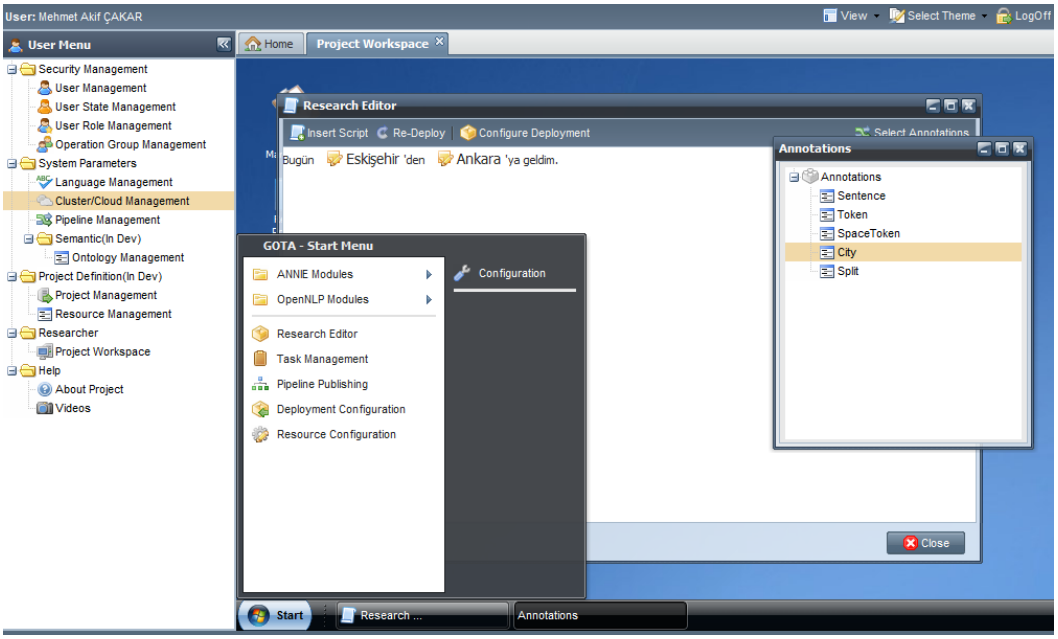
Araştırmada ilgili soruna geçici bir çözüm geliştirilmiştir. Geliştirilen çözüm, eşzamanlı çalışmalarda değişikliklerin birim zamanda kaynağa yansıtılmasıdır. Değişikliklerin kaynaklara yansıtılması sırasında, veritabanı bazında kanal kilidi kullanılmış bu şekilde birim zamanda birden fazla kullanıcının aynı kaynak üzerinde değişiklik yapması önlenmiştir. İşletim sistemi bazlı kanal kilitlemesi yerine veritabanı bazlı kanal kilitleme yolunun tercih edilme nedeni uygulamada gözetilen yatay ölçeklenebilme ilkesidir.

Önerilen yöntem, gerek birim testlerinde gerekse de pratik kullanımda herhangi bir soruna yol açmasa da kullanıcıların aynı anda aynı kaynak üzerinde farklı değişiklikler yapıp, yapılan değişiklik doğru ise sisteme yansıtması gerekliliğini sağlamamaktadır. Gerçekleştirilen çözümde eşzamanlı çalışan kullanıcılardan birinin yaptığı olası hata sonucu projenin doğru çalışması bozulduğunda, ilgili projede ki tüm kullanıcıların çalışma ortamı olumsuz yönde etkilenmektedir. Kaynakların senkronizasyonunda görülen bu sorun için araştırmacıların mümkün oldukça farklı kaynak türlerinde işlem yapacak şekilde organize edilmesi eğer mümkün değilse de sistemdeki kaynakların ihraç özelliği ile çıktısı alınıp masaüstü ortamın olanaklarından faydalanılması önerilmektedir.

3.3.4. Grafiksel kullanıcı arayüzü ve insan-bilgisayar etkileşimi

GOTA web tabanlı bir platform olduğundan web tarayıcıları üzerinden iletişim sunmaktadır. Bu noktada web tarayıcı uyumsuzluğu, çokça tartışılan ve çözülmesi gereken bir problem olarak ortaya çıkmaktadır. İnsan-bilgisayar etkileşimi alanında yazılım kalite gereksinimi olarak değerlendirilen bu problemin çözümü için yapılan araştırmalarda gerek içerdiği grafiksel kullanıcı arayüzü kontrollerinin zenginliği, gerekse de yapılan performans testlerinde başarılı sonuçlar elde edilmesinden ötürü Ext JS (Ext JS, Inc., 2010) teknolojisinin kullanımı uygun bulunmuştur.

Ext JS zengin internet uygulamaları için geliştirilmiş yeni bir tarayıcı-bağımsız Javascript kütüphanesidir. Yeni bir teknoloji olmasına ve çok az araştırmaya konu olmasına rağmen sektörel uygulamalardan, bilimsel araştırmalara kadar birçok alanda kullanımı yaygınlaşmaktadır. Ext JS'in insan-bilgisayar etkileşimine ve sistemin performansına katkısı test madenciliği alanındaki bir araştırmaya (Yang, Zhang, & Li, Information Search Based on Test Mining and EXTJS., 2009) da konu olmuştur. Projede Ext JS'nin özellikle asenkron operasyonlarla kullanımının kullanıcılara daha etkin erişim imkanları sunduğu gözlenmiştir. Ayrıca tüm süreçlerin sunucularda gerçekleştirilmesinin yerine istemci tarafında, yani web tarayıcısının içerisinde javascript kullanılarak gerçekleştirilmesi kullanıcılar açısından ek bir performans artırımı sağlamıştır. Şekil 3.7'de javascript yoluyla performans artırımının gerçekleştirildiği araştırma editörü gösterilmiştir. Internet Explorer 6+, FireFox 1.5+, Safari 3+, Chrome 3+ ve Opera 9+ web tarayıcılarında yapılan uyumluluk testlerinde temel bileşenlerin sorunsuz görüntülendiği tespit edilmiştir. Buradan hareketle tarayıcı bağımsızlığı açısından kullanıcılar için büyük bir kazanım elde edildiği sonucuna varılabilir.



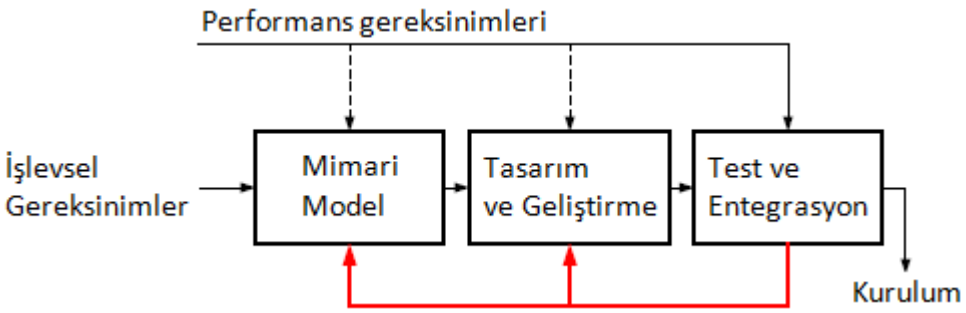
Şekil 3.7 - GOTA araştırma editörü

GKA'lerinde insan-bilgisayar arası etkileşim kalitesinin arttırılması için tamamlanması gereken diğer bir gereksinim ise engelli insanların erişim ihtiyaçlarının karşılanmasıdır. Bu noktada web tabanlı platformlarda erişilebilirlik gereksinimlerine çözüm getiren WAI-ARIA (W3C, 2008; W3C, 2010) birikimlerinden faydalanılması uygun bulunmuştur. Yapılan bir araştırmada (Mikovec, Vystrcil, & Slavik, Web toolkits accessibility study, 2009) Ext JS, Dojo (The Dojo Foundation, 2010), ICEfaces (ICESOFT, 2010) gibi popüler RIA altyapılarının erişilebilirlik açısından karşılaştırılması yapılmıştır. Araştırma sonucunda Ext JS teknolojisinin diğer RIA altyapılarına üstünlüğü tespit edilmiştir. Araştırmanın yapıldığı zamandaki Ext JS'in artılarının GOTA'nın mevcut sürümünde daha belirgin bir hal aldığı görülmüştür. Gerek engelli olan gerekse de normal kullanıcılar için ek bir iyileştirme olarak tüm girdi kontrollerinde, alan notları ve açıklamalar kullanılmıştır. Gerçekleştirilen tüm GKA etkinliklerinin GOTA platformunun kalitesine pozitif yönde ölçülebilir katkı sağladığı gözlemlenmiştir.

3.3.5. Kurulum

Yazılım süreçleri içerisinde kurulum süreci, teknolojinin üretim ortamından kullanım ortamına transferini simgeler ve aynı zamanda bir yazılım kalite kriteridir

(Haag, Raja, & Schkade, Quality function deployment usage in software development, 1996). Şekil 3.8'de temel yazılım geliştirme süreci içerisinde kurulum sürecinin yeri (Lovstad & Hughes, 2008) gösterilmektedir.



Şekil 3.8 - Temel yazılım geliştirme sürecinde kurulum (Lovstad & Hughes, 2008)

Test ve entegrasyon sürecinin başarıyla tamamlanmasının ardından başlayan ürünün son kullanıcıya sağlıklı bir biçimde transfer edilme süreci, teknolojinin sorunsuz kullanımı ile doğrudan ilgilidir. Bu nedenle kurulum süreçlerinin doğru parametrelere göre modellenmesi yazılım geliştirme süreçlerinde çözülmesi gereken problemler arasında yer almaktadır.

Kurulum süreçlerinin başarılı bir biçimde gerçekleştirilmesini negatif yönde etkileyen birçok dış parametre vardır. Proje dosyalarının hedef ortama taşınması için günümüzde birçok çözüm mevcuttur. Fakat uygulamanın çalışabilmesi için gerekli konfigürasyonların ve üçüncü parça bileşenlerin hedef ortama uygun ve doğru biçimde aktarılabilmesi oldukça sıkıntılıdır. Bu konuda yapılan bir araştırmada (Deng, 2006) kurulum ve konfigürasyon mimarilerinin tasarlanmasının zor olma nedeni olarak, tasarlanan mimarinin üçüncü parça uygulamalara, işletim sistemine veya donanıma olan bağımlılığı gösterilmiştir.

Araştırmada kurulum gereksinimleri, hedef kitleye göre iki alt başlık altında incelenebilir. Bu kitleler sırasıyla sistem yöneticileri ve son kullanıcılardır.

a) Sistem yöneticilerine dönük gereksinimler; uygulamanın merkezi bileşenlerinin hizmet sunabilir hale getirilebilmesi için gereken işlemleri içerir. Merkezi bileşenler bölüm 3.2'de değinilen ilkeler çerçevesinde geliştirilen sunuculardır. Uygulamanın dağıtık ve ölçeklenebilirliğe izin

veren yapısı, geliştirilen yazılımın aynı anda birden çok noktaya sorunsuz bir şekilde kurulabilmesini gerektirebilmektedir. Tüm bu olası ihtiyaçlara cevap verebilmek için, her sunucu türüne ait birer örnek sanal sunucu imajı hazırlanmıştır. Gerekli kurulum ve konfigürasyon işlemleri, seçilen pilot sanal sunucuda gerçekleştirilip gerekli testler tamamlandıktan sonra değişikliklerin yaması alınarak mevcut sunuculara uygulanılacak şekilde bir yaşam döngüsü belirlenmiştir. Bu yaşam döngüsüyle gerçekleştirilen birçok değişiklik sunuculara sorunsuz yansıtılmıştır.

b) Son kullanıcılara dönük gereksinimlerin karşılanması, sistem yöneticilerine dönük gereksinimlerin karşılanmasına göre oldukça zordur. Bunun başlıca nedeninin Bölüm 3.2.1'de değinilen platform bağımsızlığı ilkesi olduğu söylenebilir. Kullanıcıların farklı platformları kullanması, kurulum ve konfigürasyon sürecini olumsuz etkileyecek parametre sayısını arttırmaktadır. Bu gerekçeyle son kullanıcıya sunulan bileşenlerin alanında kendini kanıtlamış yapılar olmasına özen gösterilmiş ve son kullanıcı bileşenlerinde kurulum ve konfigürasyon ihtiyacının oldukça düşük bir seviyede tutulması amaçlanmıştır. İlgili amaç doğrultusunda GKA bileşenlerinde kullanılan Silverlight gibi üçüncü parça teknolojiler devre dışı bırakılmıştır. Sonuç olarak ortaya koyulan sistemde, son kullanıcının Ext JS teknolojisine uyumlu web tarayıcısına (ExtJS, 2010) sahip olmasının yeterli olduğu tespit edilmiştir.

3.3.6. Kültür kodlaması

Kültür kodlaması, geliştirilen sistem bileşenlerinin içeriğinden, ağda iletilen mesajlara, veritabanında saklanılan verilere kadar birçok noktada doğrudan etkili bir parametredir. Araştırmada kültür kodlamasının fonksiyonel açıdan önem arz ettiği noktaların başında dil kaynakları gelmektedir.

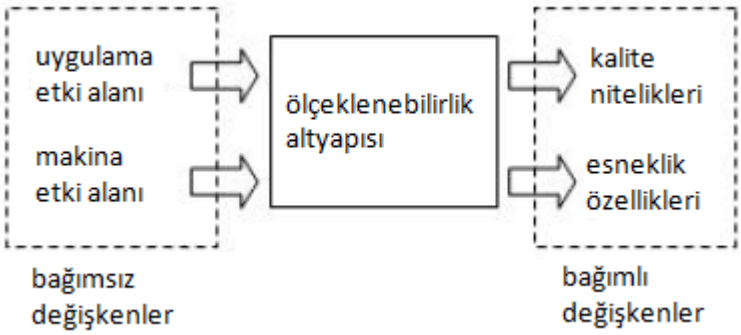
Farklı doğal dillere ait dil kaynaklarının kodlanmaları için farklı kültür kodlarının araştırmacılarca tercih edilmesi sıkça rastlanılan bir durumdur. Öte yandan son kullanıcıların işletim sistemleri, araştırma araçları gibi kullanıldıkları ortamlara ait varsayılan kültür değerleri de içeriklerin saklanma ve iletilme biçimine

etki eden faktörlerdendir. Gerek uzak noktada saklanan verinin sisteme iletilmesi gerekse de sistemde var olan verinin işlenmesi sırasında kültür kodlamasından kaynaklanan problemlere çözüm olarak ortak ve evrensel geçerli bir kodlamanın kullanılması uygun bulunmuştur. Araştırmada evrensel kodlama standartlarından (The Unicode Consortium, 2003) gerekli ihtiyaçları karşılayacak olan UTF-8 (Yergeau, 2003) kodlaması tercih edilmiştir. Bu şekilde işlenen veriler herhangi bir sorunla karşılaşmadan tek bir kodlama formatında saklanabilmektedir. Kullanıcıyla sistem arasında gerçekleşen iletişimde ise cevaplar kullanıcının talep ettiği kültür kodlamasına çevrilerek gönderilebilmektedir.

3.3.7. Ölçeklenebilirlik

Ölçeklenebilirlik paralel hesaplama (Kumar & Gupta, 1991), gereksinim önceliklendirme (Avesani, Bazzanella, Perini, & Susi, 2005), yazılım süreçleri (Laitinen, Fayad, & Ward, 2000) gibi birçok alanda tartışma konusu olmuştur. Bazı araştırmacılar (Duboc, Rosenblum, & Wicks, 2006) ölçeklenebilirliğin, geliştirilen yazılım sistemleri için birer gereksinim olarak ele alınması gerektiğini belirtilmişlerdir. Bu gerekliliğe dayanak olarakta; birçok performans faktörünün yanı sıra mesaj güvenliği, etkin bellek kullanımı ve diğer yazılım kalite kriterleri gösterilmiştir. Dolayısıyla araştırmada ölçeklenebilirliğin sağlanması, çözülmesi gereken önemli bir problem olarak ele alınmıştır.

Ölçeklenebilirliğin sağlanmasına engel olarak kalite ölçüm parametrelerinden bağımsız değişkenler gösterilebilir. Şekil 3.9'de gösterilen ölçeklenebilirlik altyapısı, bağımsız değişkenleri yazılım kalite ölçüm araçlarında parametre olarak kullanılabilecek bağımlı değişkenler olarak sunar.



Şekil 3.9- Ölçeklenebilirlik altyapısı (Duboc, Rosenblum, & Wicks, 2006)

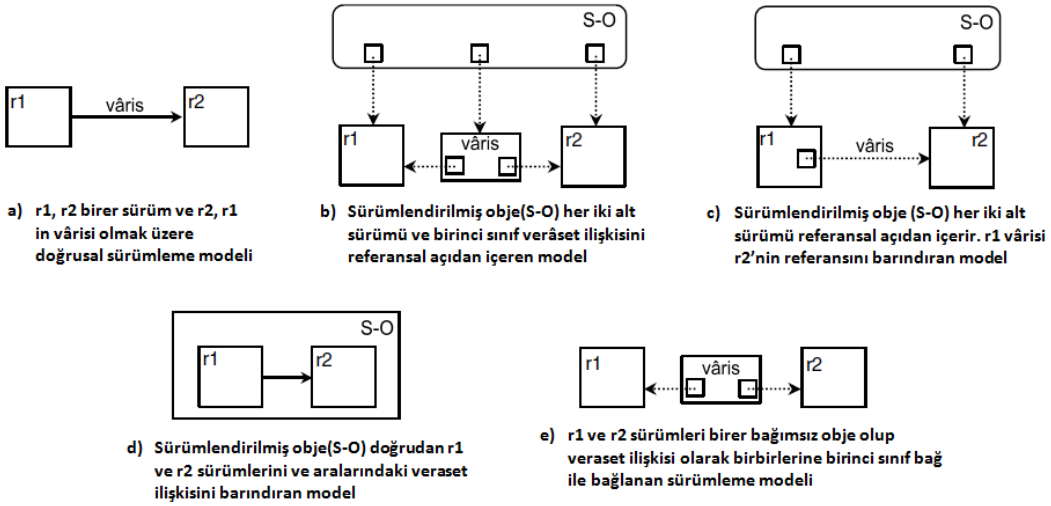
GOTA'nın ölçeklenebilirlik altyapısı, mevcut sanallaştırma (Chieu, Mohindra, Karve, & Segal, 2009) çözümlerinin kullanımına ek olarak, yazılım içerisinde çok-kanallı çalışmaya uygun bileşenlerin geliştirilmesiyle oluşturulmuştur. Sanallaştırma çözümünde mesajların ölçeklenebilirlik yönünden güvenliğinin sağlanabilmesi için mesajları işleyen sunucu, sanal bilgisayar imajı olarak çoğaltılmış ve gelen taleplerin uygulama sunucusu içerisinde yatay olarak paylaştırılması sağlanılmıştır. Bu şekilde 0.7 yatay ölçeklenebilirlik hedefine ulaşıldığı test edilmiştir. Araştırmanın modüllerinin tasarımında ise windows işletim sisteminin Monitor ve Semaphore kanallama bileşenleri kullanılarak güvenli-kanallama ilkesine dayanan bileşenler geliştirilmiştir. Bu yöntem sayesinde işlenen mesajların güvenliği işletim sistemi katmanında korunmuştur.

3.3.8. Konfigürasyon yönetimi ve sürümlendirme

Sürümlendirme ve sürümlendirme modelleri terim olarak, yazılım konfigürasyon yönetimi alanının altında incelenmekte ve projelerin geliştirilme sürecinde önemli yer tutmaktadır (Wheeler, 2005). Sürümlendirme yapılmayan bir platformda değişikliklerin izlenebilmesi oldukça zorlaşırken, kurulum sürecinde sorunlarla karşılaşılmasına da meydan vermektedir. Ayrıca geliştirilen projelerin yayımlanmasında da birçok uyumsuzluk meydana gelmektedir. Araştırmada sürümlendirme ihtiyacı üç alt başlıkta incelenmektedir;

a) Kaynak kodunun sürümlendirilmesi: Yapılan araştırmalar ve geçmiş deneyimler sonucu, kaynak kodunun sürümlendirme ihtiyacının karşılanması için Subversion (Nagel, 2005) kullanımı uygun bulunmuştur.
b) Bileşenlerin sürümlendirilmesi: Dağıtık mimariyle tasarlanan uygulamaların yayımlanması sırasında değişen bir bileşenin kullandığı kaynak bir başka bileşen ile ortak paylaşıldığı durumlarda uyumsuzluk sorunlarıyla karşılaşılmaktadır. Uyumsuzluk sorununa yönelik çözüm olarak, servis tabanlı mimari kullanımının yanı sıra aynı ortamda paylaşılması performans açışından zorunlu bileşenler için ilgili araştırmada (Stuckenholz, 2005) önerilen konfigürasyon aracıyla sürümleme politikasına doğrudan müdahale yöntemi benimsenmiştir.
c) Kaynakların sürümlendirilmesi: Kaynak sürümü(r) sistemde mevcut bir kaynak üzerinde yatay zaman diliminde meydana gelen değişimi temsil eder. Bir sürümün, sürümlendirme işaretçisiyle hangi yolla temsil edileceği, hangi karakteristiklerin sürüm hesaplamaya etki edeceği araştırmacılar (Conradi & Westfechtel, 1998) için inceleme konusu olmuştur.

Araştırma ortamına uygun sürümlendirme modelini belirlemek, mevcut sürümlendirme modellerinin avantaj ve dezavantajları saptamak için yapılan araştırma sürecinde Şekil 3.10'de gösterilen durum bazlı modeller incelenmiştir. İnceleme sonucunda araştırma ortamında bulunan dil ve işleme kaynaklarının gerek veri tipleri gerekse de içerik karakteristikleri nedeniyle sürümler arasındaki ilişkinin doğrudan tutulması sakıncalı bulunmuştur. Bu nedenle alt sürümlerin, sürümlendirilmiş objeler(S-O) aracılığıyla temsil edilmesi uygun bulunmuştur. r1, r2 gibi alt sürümlerin arasında da birinci sınıf referansal ilişki modeli kullanılmıştır.



a) r1, r2 birer sürüm ve r2, r1 in vârisi olmak üzere doğrusal sürümleme modeli

b) Sürümlendirilmiş obje(S-O) her iki alt sürümü ve birinci sınıf verâset ilişkisini referansal açıdan içeren model

c) Sürümlendirilmiş obje (S-O) her iki alt sürümü referansal açıdan içerir. r1 vârisi r2'nin referansını barındıran model

d) Sürümlendirilmiş obje(S-O) doğrudan r1 ve r2 sürümlerini ve aralarındaki veraset ilişkisini barındıran model

e) r1 ve r2 sürümleri birer bağımsız obje olup veraset ilişkisi olarak birbirlerine birinci sınıf bağ ile bağlanan sürümleme modeli

Şekil 3.10 - Objenin sürümleme geçmişi için seçilebilecek durum bazlı modeller (Whitehead, 2001)

r1 ve r2 doğrudan temsil edilmediği için çalışmanın güncel sürümünde doğrudan sürümler arası geçiş sağlanamamaktadır. Araştırmacının gerektiğinde bileşenler arasında geçmiş tarihe dönememesi konfigürasyon yönetiminin dönülebilirlik ilkesini yerine getirmediğinden bu alanda bir eksik olduğu saptanmıştır.

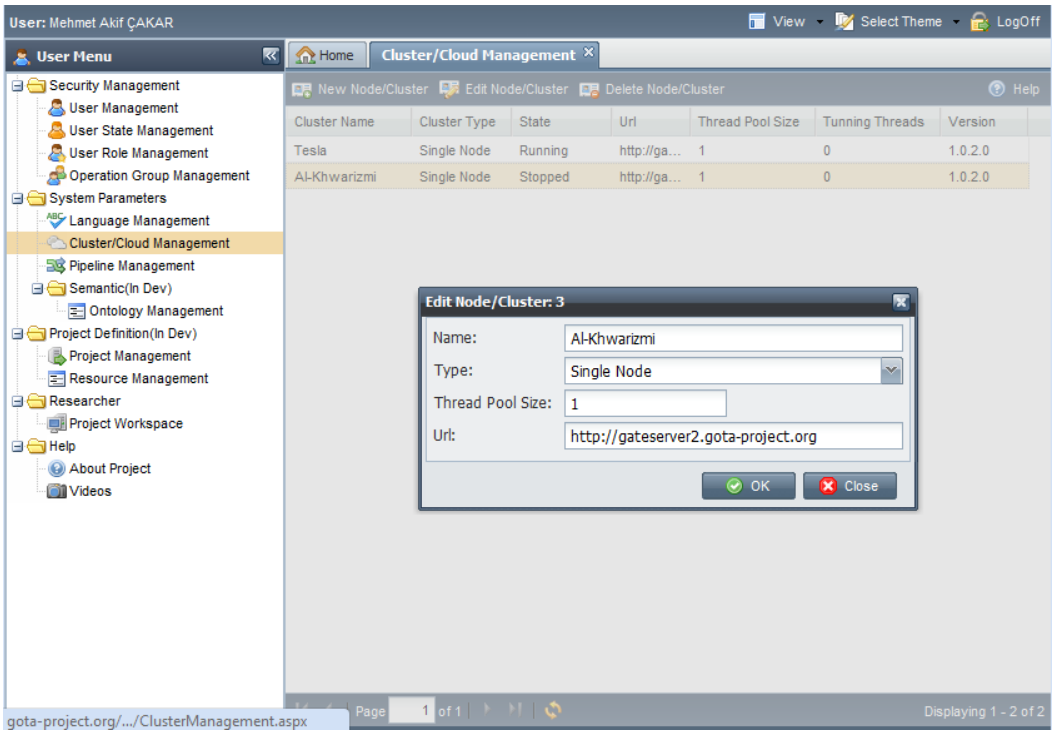
3.3.9. Performans

GOTA projesinin kullanıma açılmasından önce yapılan ölçeklenebilirlik testlerinde performans ihtiyacı ortaya çıkmıştır. Projede ilgili performans ihtiyacını gidermek için birçok teknoloji kullanılmıştır. Kullanılan teknolojilerin başında, projeye ait statik verilerin içerik dağıtım ağları üzerinden yayımlanması gelmektedir. Bu şekilde GOTA sunucularının, üzerinde olması gereken işlemsel yükünün dışında kalan statik veri iletişim yükünden arındırılması amaçlanmıştır. Bu amaç doğrultusunda yapılan araştırmalarda QoS mekanizmasını kullanan İDA'lar üzerinden içerik dağıtım sürecinin gerçekleştirilmesi uygun görülmüştür. QoS mekanizmasının İDA üzerindeki verimi çeşitli araştırmalarda (Pathan & Buyya, 2009) incelenmiştir.

Statik içerik için var olan İDA'lardan uygun bir çözüm seçildikten sonra projeye ait alt bir alan(cdn.gota-project.org) ilgili ağa yönlendirilmiştir. İçerik dağıtım ağları ve performans artırım metodolojilerinin detaylı olarak incelendiği bir

çalışmada (Hull, Content Delivery Networks: Web Switching for Security, Availability, and Speed, 2002) ele alınan birçok metodolojide GOTA platformuna uygulanmıştır. Yapılan işlem sonucunda sistemin mevcut ve geçmiş durumları arasında yapılan performans testinin sonuçları Bölüm 3.4.2'te gösterilmiştir.

Performans artırımında tercih edilen bir diğer yöntem ise sanallaştırma yoluyla yatay ölçeklendirmedir (Trieu, Ajay, Alexei, & Alla, 2009). Genel anlamda uygulama katmanında gerçekleştirilen ölçeklendirme yöntemlerinin performansa yönelik etkinliği çeşitli çalışmalarda (Tan, Waters, & Crawford, 2003) incelenmiştir. Araştırmada ağ katmanında ve uygulama katmanında olmak üzere iki ayrı ölçekleme metodu kullanılmıştır. Ağ katmanında kümelemeye dayalı (Lamb , Windows 2000 Clustering and Load Balancing Handbook, 2001) üçüncü seviye yazılımsal ağ yük dağıtıcısı(ing. Netwok Load Balancer) kullanılmıştır. Uygulama seviyesinde ise gelen talepleri sistemde tanımlı işlemcilere vektörel dağıtım yapan bir servis (Şekil 3.11) geliştirilmiştir. Bu şekilde ağ katmanına ek olarak bir ölçeklendirme derecesi elde edilmiştir. Geliştirilen bu yapıya ait stres testinin sonuçları Bölüm 3.4.2'de gösterilmiştir.



Şekil 3.11 - Kümeleme ayarlarını düzenleme ekranı

Uzak yerleşkedeki kullanıcıların sistemle iletişimleri internet ağ üzerinden gerçekleştirildiği için sunucudan çıkan yanıtların sıkıştırılması bir başka performans optimizasyonu olarak ele alınmıştır. GZip (Deutsch, RFC1952: GZIP file format specification sürüm 4.3, 1996) ve Deflate (Deutsch, RFC1951: DEFLATE Compressed Data Format Specification sürüm 1.3, 1996) algoritmaları istemciyle sunucu arasındaki iletişimde standart olarak kabul görülmüş veri sıkıştırma algoritmalarıdır (King, Website optimization, 2008). Microsoft IIS 7.0 web sunucusuyla birlikte gelen sabit ve değişken içeriklerin otomatik olarak sıkıştırılma özelliği (Mueller, 2007) gerekli ayarlar ile aktifleştirildikten sonra elde edilen sonuçlar Bölüm 3.4.2'de gösterilmiştir.

Gerçekleştirilen tüm bu performans optimizasyonlarının dışında sitil tanımlarının HTML (WWWC, 2010) sayfaların başlık kısmına, javascript dosyalarının ise sayfaların alt kısmına alınması gibi birçok operasyon gerçekleştirilmiştir. Fakat yapılan testlerde gerçekleştirilen ek optimizasyonların anlamlı bir fark sağlamadığı anlaşıldığından araştırma kapsamı dışında tutulmuştur.

3.3.10. Doğrulama ve yetkilendirme

Günümüzde temel doğrulama mantığıyla (Burrows, Abadi, & Needham, 1990) uygulanılabilen yöntemler, birçok programlama altyapısı ve uygulama sunucusuyla bütünleşik olarak sunulmaktadır. Fakat yetkilendirme yöntemleri sistemlerin gereksinimlerine göre sürekli gelişmek zorundadırlar. GOTA projesinde, kullanıcıların işbirliği içerisinde çalışabilmelerine imkân ve ileride karşılaşılabilecek dinamik rol gereksinimlerine cevap verebilecek bir yetkilendirme mekanizması gerekliliği saptanmıştır.

GOTA platformunun dağıtık programlama mimarisi aynı zamanda dağıtık yetkilendirme ihtiyacını doğurmuştur. Bu ihtiyaca yönelik yapılan araştırmalar (Woo & Lam, 1993; Avijit, Datta, & Harper, 2010) incelenmiş olup önerilen mekanizma genel anlamda sistemin geliştirilmesinde model olarak kullanılmıştır.

Araştırmada geliştirilen yetkilendirme mimarisi kullanıcının haricinde dört ayrı katmandan oluşmaktadır. Şekil 3.12'de sırasıyla yetkilendirme katmanları arasındaki ilişki gösterilmektedir. Katmanlar arası ilişki düzeyi, en üst katmanda kullanıcı olmak üzere aşağıya doğru bire-çok ilişki modeli çerçevesine dayanır.

Şekil 3.12 - Yetkilendirme katmanları

En alt seviye olan beşinci seviyeden yukarıya doğru bu katmanları incelemek gerekirse;

a) 5. Seviye - Tanımlamalar ve Olaylar: Sistemde gerçekleşebilecek her bir atomik süreç tanım olarak nitelendirilir. Olay ise sistemde gerçekleşebilecek önceden tanımlanmış standart hareketleri temsil etmektedir. Araştırmanın 0.1.1.4 sürümü itibariyle altı temel olay, gerekli tüm ihtiyaçları karşılamıştır. Bu olaylar sırasıyla oluşturma, değiştirme, okuma, silme, arama ve görüntülemedir. Beşinci seviyenin amacı, yetkilendirme mekanizmasıyla yazılım kodu arasındaki soyutlamayı gerçekleyen anahtar katman görevi görmektir. Yalnızca bu seviyede yapılan değişiklikler uygulama koduna doğrudan etki etmektedir. Bu katmanda olaylar ve tanımlar arasında herhangi bir ilişki yoktur. Kod içerisinde kullanıcının yetkisi sorgulanmak istendiğinde iki tür soru sorulabilir;

i. Yetki gerektirir: Herhangi bir sınıf veya metodun başına nitelik olarak eklenen soru sorma şeklidir. Çizelge 3.1'de yetki gerektirir tanımlama şeklinin uygulama içerisindeki örnek kullanımı gösterilmektedir. Örnekte;

RequiresGOTAPermission nitelikleri DoDelete metodunun başına eklenmiştir. DoDelete metodu herhangi bir yerden çağrıldığında, içerik kodu uygulanmadan önce kendisine ait tüm yetki gerektirir tanımlamaları denetlenir. İlgili tanım ve olaylara kullanıcının yetkisi yoksa metot yetki hatası fırlatır. Benzer şekilde yetki gerektirir tanımlamaları sınıfların başına tanımlandığında, ilgili sınıflar bellekte oluşturulmadan önce gerekli denetim işlemleri yapılır. Bu tanımlama şekli kodun karmaşıklık çarpanını düşmekte ve okunurluğu artırmaktadır. Visual Studio Code Metrics (Microsoft, Code Metrics Overview, 2008) ile yapılan ölçümlerde kod karmaşıklığının yaklaşık %11 ile %27 arasında azaldığını görülmüştür. Yapılan ölçümler kod örüntüsünün karmaşıklık çarpanına göre artış veya azalış gösterebileceğinden kesin bir oran/orantı belirtmek sakıncalı görülmüştür.

Çizelge 3.1 - Yetki gerektirir tanımlama türü kod örneklemi

```
[RequiresGOTAPermission(GOTADefinition.PublishedPipelinesTable, EventsEnum.Delete)]
[RequiresGOTAPermission(GOTADefinition.PublishedPipelinesTable, EventsEnum.Read)]
protected override void DoDelete(string sectionId, string key, DirectEventArgs e)
{
   using (var context = ContextManager.NewGOTADataContext)
   {
      var item = context.PublishedPipelines.Single(o => o.Id == int.Parse(key));
      context.PublishedPipelines.DeleteOnSubmit(item);
      context.SubmitChanges();
   }
   crudInfo.Reload();
}
```

ii. Yetkili Mi?: Kod örüntüsü içerisinde yetkiye göre dallanma ihtiyacı gerektiğinde kullanılabilecek soru şeklidir. Bu soru şekli de yetki gerektirir soru şekli gibi kullanıcının ilgili tanıma ve olaya yetkilisi olup olmadığını sorgular.

Her iki soru tipinde de soru içeriği tanım ve olay ile kısıtlandırılmıştır. Rol tabanlı yöntemlerde (Ahn & Sandhu, 2000), (Yao & Tamassia, 2009) olduğu gibi kullanıcının rolü veya yetkisi üzerinden soru sorma şekli kesin bir biçimde reddedilmiştir.

b) 4. Seviye – Operasyonlar: Operasyonlar beşinci seviyeye ait tanım ve olayların, çiftler halinde ilişkilendirilerek temsil edilmesi için tasarlanmıştır. Sistemde soruları operasyonlar aracılığıyla sormak yerine, operasyonları tanım ve olay şeklinde iki ayrı gruba ayırarak sormanın birtakım nedenleri vardır. Bu nedenlerin başlıcaları sistemde geri izleme sırasında tanım ve/veya olay bazlı arama yapabilmek ve veritabanı tasarımında en alt seviye olarak üçüncü normal form (Kent, 1989) ilkelerinin hedeflenmesidir.

c) 3. Seviye – Operasyon Grupları: Dördüncü seviyede tanımlanan operasyonları gruplar halinde ifade edebilmeye yarayan mantıksal katmandır. Örneğin; proje oluşturmak için gerekli operasyonlar ProjeTablosu-Oluşturma, EklentiTablosu-Okuma, KullanıcıTablosu-Arama gibi birçok farklı tanım ve olay çiftinden oluşmaktadır. İşlevsel ve mantıksal olarak operasyonları kümelemek için üçüncü seviye kullanılır. Bu şekilde rol düzenleme sürecinde atomik tanımlamalar yerine işlevsel tanımlamalar üzerinden işlem yapabilme imkânı tanınmıştır.

d) 2. Seviye – Roller: Kullanıcının organizasyondaki konumunu temsil etmek için tasarlanan mantıksal katmandır. Operasyon gruplarından sonra benzer amaca hizmet eden yeni bir katmanın tasarlanmasının nedeni sistem yöneticileri için anlaşılabilir adlandırmalar sağlamaktır. Örneğin araştırmada, sistem yöneticisi, proje yöneticisi, araştırmacı, anonim kullanıcı gibi ön tanımlı roller bulunmaktadır.

e) 1. Seviye – Kullanıcılar: Sistemde tanımlı gerçek ve/veya temsili kişilerdir. Sisteme giriş kullanıcıya ait güvenlik jetonları(ing. security token) (Matsuura, 2003) kullanılarak yapılmaktadır. Temel olarak kullanıcı adı/parola çifti ile sayısal sertifika türünü destekleyen yetkilendirme bileşenleri doğrudan kullanıcı seviyesiyle ilişkilidir.

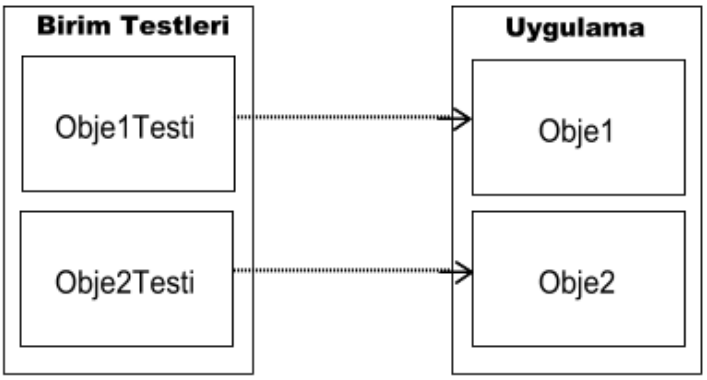
3.4. Yapılan Testler ve Sonuçları

GOTA platformunda önceki bölümlerde bahsedilen problem ve çözüm önerilerine yönelik yapılan faaliyetlerin öncesinde ve sonrasında birtakım testler gerçekleştirilmiştir. Yapılan testler, araştırmanın amaç ve işlevleri doğrultusunda yapılan testler ile performans ve ölçeklenebilirliğe yönelik testler olarak iki alt başlık altında toplanmaktadır.

3.4.1. İşlevsel testler

Birim testleri yazılım mühendisliğinin anahtar bileşenlerindendir (Tillmann & Schulte, 2005). Önemli bir bileşen olmasından ötürü, yazılım firmaları ve topluluklar birim testlerini geliştirebilmek için birçok araç, altyapı ve servisler yayınlamışlardır.

GOTA platformunun fonksiyonalitesini doğrudan ilgilendiren konularda birtakım testlerin yapılması gerekmiştir. Araştırma sürecinde ortaya çıkan bileşenler için doğrudan birim test uygulanmıştır. Üçüncü parça bağımsız bileşenler içinse ek bir çalışma yapılmamıştır. GOTA, temelde iki farklı programlama dili kullanılarak geliştirildiği için iki farklı birim test aracına ihtiyaç duyulmuştur. Java dilinde yazılan modüller için JTest (Parasoft, 2009) ve JUnit (Beck, 2004), C#.Net diliyle yazılan modüller içinse NUnit (Hamilton, 2004) kullanılmıştır. Yazılan birim testleri uygulamadan bağımsız olarak tasarlanmalıdırlar (Hamill, 2004) ilkesinden hareketle araştırmada bağımsız tasarım ilkesi(Şekil 3.13) gözetilmiştir.



Şekil 3.13 - Birim Testleri ve Uygulama (Hamill, 2004)

Geliştirilen birim testleri uygulamanın yayınlanmasından önce periyodik olarak çalıştırılarak kod denetim aracı olarak kullanılmıştır.

3.4.2. Yapısal testler

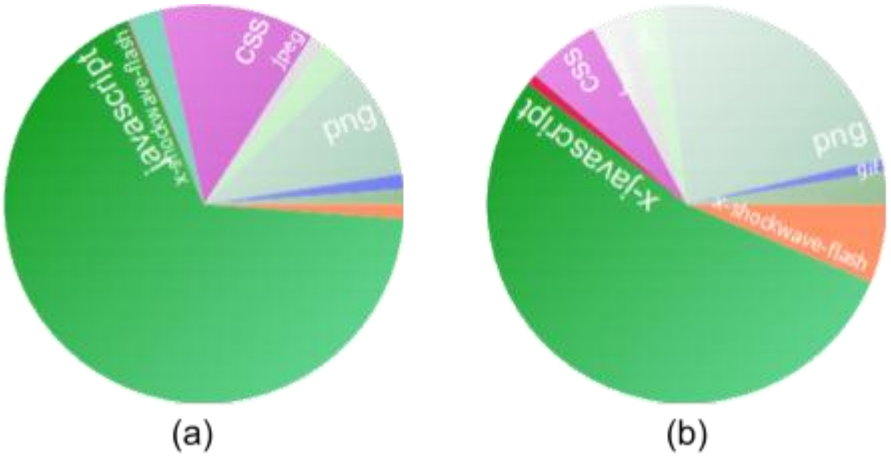
Sistemin geliştirilme sürecinde denetim mekanizması olarak tasarlanan yapısal testlere ek olarak yayınlanma aşamasından öncede bölüm 3.3.9'da değinilen performans testleri gerçekleştirilmiştir. Karşılaştırmalı tüm testlerde t1 sistemin yalın halini t2 ise sistemde gerekli optimizasyon çalışması yapıldıktan sonraki halini gösterir. Gerçekleştirilen testler sırayla ele alınırsa;

a) Gzip ve Deflate Sıkıştırma Testi: Web sitelerinin kullanıcılar tarafından daha hızlı açılabilmeleri için gerekli performans optimizasyonu çalışmalarının sonucunda yapılan testin içerik türüne göre sonuçları Çizelge 3.1'de görülmektedir.

Çizelge 3.2 - İçerik türüne göre sıkıştırma sonuçları

	t1	t2	Kazanım(%)
~http başlıkları	16,579	17,293	-%4
image/x-icon	16,022	16,022	%0
image/jpeg	33,749	34,247	-%1
image/png	132,727	132,727	%0
image/gif	16,739	16,739	%0
text/css	169,517	44,265	%74
text/javascript	895,779	330,772	%63
text/html	16,084	6,079	%62
application/x-javascript	15,301	14,557	%5
Diğer	38,331	38,329	%0
Toplam Alınan Veri(byte)	1,350,828	651,030	%52

Gzip sıkıştırma süreci sonrasında toplamda %52 ye varan kazanç sağlanmıştır. Toplam kazanımın içerik türüne göre dağılımı Şekil 3.14'te gösterilmektedir.



Şekil 3.14 - Testin içeriklere göre dağılımı (a) t1 durumundaki dağılım (b) t2 durumundaki dağılımı göstermektedir.

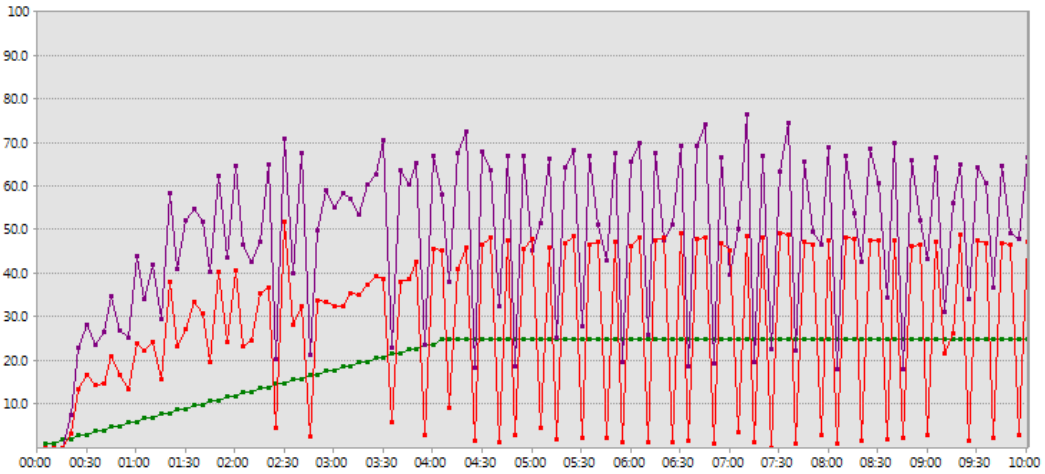
Javascript, stil tanımları, html içerikler gibi metin türündeki verilerde anlamlı bir kazanım sağlanmıştır. Sonuç olarak elde edilen kazanım sonucunda uygulamaya ulaşmak isteyen kullanıcıların normalin yarısı kadar bir sürede içeriğe erişebilmeleri sağlanmıştır.

b) Stres ve Yük Testi: Yazılım testleri içerisinde önemli bir yer tutan stres ve yük testleri (Beizer, 1984; Binder, 2000), sistemin olası yüklenme karşısında kararlılığını ve destekleyebileceği kullanıcı miktarını tespit etmek için kullanılır. Araştırmada, sık kullanılan bir kullanıcı senaryosu Visual Studio Team System (Microsoft, Microsoft Visual Studio Team System, 2010) kullanılarak kaydedilmiştir. Kaydedilen senaryo, 10 kullanıcıdan başlayarak periyodik olarak 250 kullanıcıya kadar artan kullanıcı yüküyle çalıştırılmıştır. Test parametrelerinde bağlantı havuzu kullanılmamıştır. Senaryonun t1 ve t2'de uygulanmasının sonucu Çizelge 3.3'te gösterilmiştir.

Çizelge 3.3 - Stres ve yük testinin sonuçları

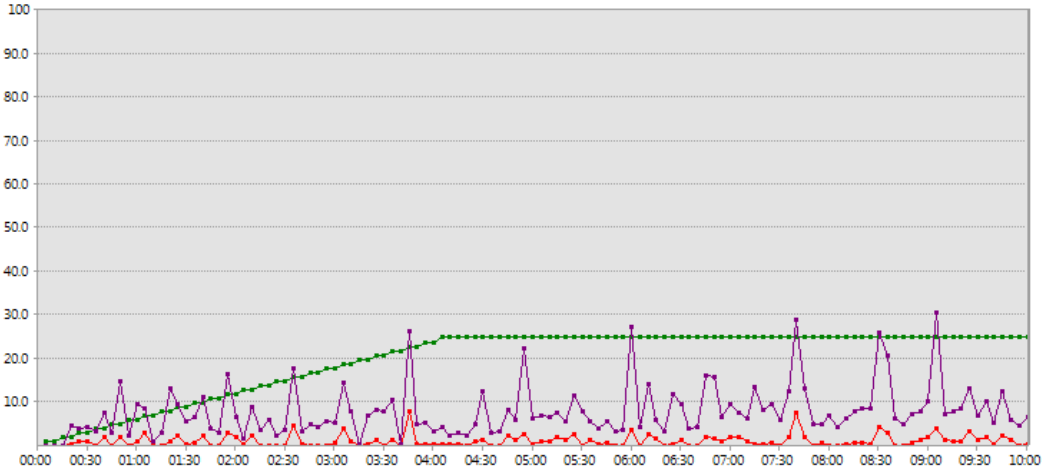
	t1		t2	
	En Yüksek	Ortalama	En Yüksek	Ortalama
Kullanıcı yükü	250	200	250	200
Yanıt Süresi	768	485	30,6	8,4
Hata Sayısı	52,2	28,5	8	1,23

Yapılan optimizasyonlar öncesi ortalama 200 kullanıcı için 28,5 talep hata alırken ortalama yanıt süresi 485 birim zamandır. Değişiklikler sayesinde ortalama 200 kullanıcı için 1,23 talep hata alırken ortalama yanıt süresi 8 birim zamana düşmüştür. İlgili sonuçlar oranlandığında hatalarda %96 oranında bir azalma görülürken yanıt süresinde %98 oranında azalma elde edilmiştir. Sonuç değerlerinin test sürecindeki dağılımı t1 için Şekil 3.15, t2 içinse Şekil 3.16'te gösterilmektedir.



Şekil 3.15 - Optimizasyon işlemleri öncesi yapılan stres testi çıktı grafiği

Optimizasyon öncesi uygulamada elde edilen değerlerde sıklıkla dalgalanmalar görülmektedir. Meydana gelen değişim aralığının zaman zaman 50 birime ulaştığı gözlenmiştir.



Şekil 3.16 - Optimizasyon işlemleri sonrası yapılan stres testi çıktı grafiği

Optimizasyon sonrası uygulamada elde edilen değerlerde görülen dalgalanmalar 26 birimi geçmemekle birlikte stres testlerinde beklenilen bir seyir izlemekte olduğu görülmüştür.

4. SONUÇ VE ÖNERİLER

Bu çalışmada, tasarlanan GOTA kod adlı mimariye dayalı web tabanlı DDİ platformunun geliştirilme sürecinde elde edilen kazanımlar ortaya koyulmuştur. Geliştirilme sürecinde karşılaşılan farklı disiplinlere ait birçok probleme çözüm sunulmuştur. Elde edilen kazanımlar, araştırma dünyasının ihtiyaçlarına ve sektörel gereksinimlere göre gruplandırılırsa;

- Araştırmacılar için; takım halinde DDİ projelerini geliştirebilme imkânı ortaya koyulmuştur. Proje ekibinde yetkilendirmeye imkân sağlanmış araştırmacılar arası görevlerin yönetimi için özel modüller geliştirilmiştir. Araştırmacılara farklı DDİ araçları ve altyapılarını kullanarak hibrit projeler geliştirebilme olanağı sunulmuştur. Platform üzerinde geliştirilen projeleri BNEP web servisi, e-posta gibi birçok kanalla dış dünya ya tek tıklamayla yayınlayabilmek mümkündür. GKA alanında Ext JS kullanımıyla birlikte web tabanlı platformlarda görülen web tarayıcı uyumsuzluğu gibi birçok sorun giderilmiştir. İnsan-bilgisayar etkileşimi alanında WAI-ARIA standartları gözetilmiş bu sayede son kullanıcılara daha kolay kullanılır arabirimler sunulmuştur.

- Sektörel kullanım için; GOTA üzerinde geliştirilen DDİ projelerinin istenilen aşamasına servis tabanlı mimari sayesinde doğrudan entegre olabilme imkânı sunulmuştur. Platform üzerinde birçok yazılım kalite ölçüm testi uygulanmış ve bu testler sonucunda sistemin ölçeklenebilirlik ve güvenilirlik açısından başarılı bir sonuçlar verdiği görülmüştür. GOTA platformunda kullanılan dağıtık bileşen mimarisinin; ileride ihtiyaç duyulabilecek üçüncü parça DDİ araçlarının sisteme kolayca entegre edilebilmesine imkân verdiği görülmüştür. Sistemin birçok yönden optimize edilmesi sonucu maliyet açısından etkin bir platformun elde edildiği söylenilebilir. Ayrıca GOTA'nın açık kaynak kodlu bir platform olması da sektörel uygulamalarda maliyeti düşüren bir etken olarak değerlendirilebilir.

Elde edilen tüm bu kazanımların yanı sıra GOTA platformuna paralel programlama ve bulut-bilişim desteğinin getirilmesi için çalışmalara başlanılmış ve bu konu üzerinde birkaç ön birim-testi gerçekleştirilmiştir. Fakat gerekli tüm testler tamamlanması için gerekli şartlar oluşturulamadığından bu konular tez çalışmasının kapsamı dışında tutulmuştur.

Platformda gelecekte desteklenmesi istenilen hedeflere değinmek gerekirse;

- Sürümleme ve kaynakların senkronizasyonu problemine daha etkin bir çözümün geliştirilmesi gerekmektedir.
- UIMA, LingPipe gibi araştırma dünyasında kabul görmüş DDİ araçlarına uygun adaptörler ve GKA'ları geliştirilerek GOTA platformuna entegre edilmesi hedeflenmektedir.
- Bulut-bilişim desteği verilebilecek bileşenlerin, mimariye uygun yapıda geliştirilmesine ve/veya entegre edilmesine ihtiyaç duyulmaktadır.
- Geliştirilen DDİ projelerinin doğrudan web sitelerinde bulunan içerikler üzerinde çalıştırılabilmesi için var olan web tarayıcı araçlarından uygun olanın belirlenip GOTA platformuna yeni bir bileşen türü olarak eklenmesi hedeflenmektedir.
- Bilgi çıkarımı sonuçlarının ontolojilerle eşleştirilmesi için gereken bileşen türlerine destek verilmesine ihtiyaç duyulmaktadır.
- Web tabanlı araştırma ortamına ek olarak Netbeans, Visual Studio gibi popüler yazılım geliştirme ortamları içerisinde GOTA projelerinin geliştirilebilmesi için gerekli bileşen ve proje şablonlarının geliştirilmesi hedeflenmektedir.

5. KAYNAKLAR

ICESOFT. (2010). *ICEfaces web toolkit*. Mayıs 15, 2010 tarihinde http://www.icefaces.org/ adresinden alındı

Ahn, G.-J., & Sandhu, R. (2000). Role-based authorization constraints specification. *ACM Transactions on Information and System Security (TISSEC)*, 207-226.

Alias-i. (2008). *LingPipe 3.9.2*. May 2010 tarihinde http://alias-i.com/lingpipe adresinden alındı

Avesani, P., Bazzanella, C., Perini, A., & Susi, A. (2005). Facing Scalability Issues in Requirements Prioritization with Machine Learning Techniques. *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, 297-306.

Avijit, K., Datta, A., & Harper, R. (2010). Distributed programming with distributed authorization. *Types In Languages Design And Implementation*, 27-38.

Baqai, S. (1998). *Resource allocation and synchronization protocols for quality-based presentation in distributed multimedia systems.* West Lafayette, IN, USA: Purdue University.

Beck, K. (2004). *JUnit Pocket Guide.* Sebastopol, CA, USA: O'REILLY.

Beizer, B. (1984). *Software system testing and quality assurance.* Pennsauken, NJ: Van Nostrand Reinhold Electrical/Computer Science And Engineering Series.

Binder, R. (2000). *Testing Object-Oriented Systems - Models, Patterns, and Tools.* Addison-Wesley.

Burrows, M., Abadi, M., & Needham, R. (1990). A logic of authentication. *ACM Transactions on Computer Systems (TOCS)*, 18 - 36.

Chappell, D. (2004). *Enterprise Service Bus.* O'Reilly Media, Inc.

Chappell, D. (2006, October). *Service Reuse -- Fact or Fiction?* May 2010 tarihinde O'Reilly XML.COM: http://www.oreillynet.com/xml/blog/2006/10/service_reuse_fact_or_fiction.html adresinden alındı

Chieu, T., Mohindra, A., Karve, A., & Segal, A. (2009). Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. *Proceedings of the 2009 IEEE International Conference on e-Business Engineering*, 281-286.

Clarke, L. (1996). How do we improve software quality and how do we show that it matters? *ACM Computing Surveys (CSUR)*, Article No.: 203.

Conradi, R., & Westfechtel, B. (1998). Version models for software configuration management. *ACM Computing Surveys (CSUR)*, 232-282.

Cunningham, H. (2000). *Software Architecture for Language Engineering.* Sheffield, UK: Doktora Tezi, University of Sheffield.

Cunningham, H., Maynard, D., Bontcheva, K., & Tablan, V. (2002, July). GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02).* Philadelphia.

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Dimitrov, M., Dowman, M., et al. (2009). *Developing Language Processing Components with GATE*. 3 9, 2010 tarihinde GATE User Guide: http://gate.ac.uk/userguide/ adresinden alındı

Deng, G. (2006). Resolving component deployment & configuration challenges for enterprise DRE systems via frameworks & generative techniques. *Proceedings of the 28th international conference on Software engineering*, 945-948.

Deutsch, P. (1996). RFC1951: DEFLATE Compressed Data Format Specification sürüm 1.3. USA: RFC Editor.

Deutsch, P. (1996). RFC1952: GZIP file format specification sürüm 4.3. USA: Internet RFCs.

Dooley, K. (2002). Designing large-scale LANs. O'Reilly & Associates, Inc.

Duboc, L., Rosenblum, D., & Wicks, T. (2006). A framework for modelling and analysis of software systems scalability. *Proceedings of the 28th international conference on Software engineering*, 949-952.

Eryiğit, G. (2007). ITU treebank annotation tool. (s. 117-120). Prague, Czech Republic: Association for Computational Linguistics.

Ext JS, Inc. (2010). *ExtJS Ürün Tanımı*. Mayıs 25, 2010 tarihinde ExtJS Web Sitesi: http://www.extjs.com/products/js/ adresinden alındı

ExtJS. (2010). *Ext Core: Cross-Browser Javascript Library*. 5 1, 2010 tarihinde ExtJS Community: http://www.extjs.com/products/core/ adresinden alındı

Fisher, M., Lai, R., Sharma, S., & Moroney, L. (2006). *Java EE and .NET Interoperability: Integration Strategies, Patterns, and Best Practices.* Upper Saddle River, NJ, USA: Prentice Hall PTR.

Fricko, A. (2006, May). *SOAs Require Culture Change And Service Reuse.* May 2010 tarihinde Business Communications Review: http://www.allbusiness.com/business_planning/business_structures/3603441--1.html adresinden alındı

Grefenstette, G. (1998). *Cross-Language Information Retrieval.* Norwell MA, USA: Kluwer Academic Publishers.

Grimes, S. (2006). *Unstructured Data and the 80 Percent Rule*. Nisan 17, 2010 tarihinde http://clarabridge.com/default.aspx?tabid=137&ModuleID=635&ArticleID=551 adresinden alındı

Haag, S., & Hogan, P. (1992). Research issues in software quality function deployment: A new beginning for software engineering methodologies. *roceedings of Decision Sciences Institute*, 926–928.

Haag, S., Raja, M., & Schkade, L. (1996). Quality function deployment usage in software development. *Communications of the ACM*, 41-49.

Hamill, P. (2004). *Unit test frameworks.* O'Reilly.

Hamilton, B. (2004). *NUnit Pocket Reference.* Sebastopol, CA, USA: O'REILLY.

Hull, S. (2002). *Content Delivery Networks: Web Switching for Security, Availability, and Speed.* New York, NY, USA: McGraw-Hill, Inc.

Jordi, T., Ageno, A., & Català, N. (2006). Adaptive information extraction. *ACM Computing Surveys (CSUR)* (s. 2-4). New York, NY, USA: ACM.

Keith, R. (2005, September 19). *Profiting from service reuse.* May 2010 tarihinde http://www.looselycoupled.com/stories/2005/rearden-ca0919.html adresinden alındı

Kent, W. (1989). A simple guide to five normal forms in relational database theory. *Parallel architectures for database systems*, 66-71.

King, A. (2008). *Website optimization.* O'Reilly .

Kumar, V., & Gupta, A. (1991). Analysis of scalability of parallel algorithms and architectures: a survey. *International Conference on Supercomputing*, 396-405.

Laitinen, M., Fayad, M., & Ward, R. (2000). Thinking Objectively: The problem with scalability. *Communications of the ACM*, 105-107.

Lamb , J. (2001). *Windows 2000 Clustering and Load Balancing Handbook.* Upper Saddle River, NJ, USA: Prentice Hall PTR.

Lovstad, J., & Hughes, P. (2008). Run-time software configuration for mobile devices using an evolutionary quantifiable deployment model. *Proceedings of the 7th international workshop on Software and performance*, 189-200.

Matsuura, K. (2003). Digital Security Tokens and Their Derivatives. *Kluwer Academic Publishers*, 161-179.

Maynard, D., Cunningham, H., Botcheva, K., Catizone, R., Demetriou, G., Gaizauskas, R., et al. (2000). *A Survey of Uses of GATE.* UK: Department of Computer Science, University of Sheffield.

McKendrick, J. (2006, August). *Pouring cold water on SOA reuse' mantra.* May 2010 tarihinde ZDNet: http://www.zdnet.com/blog/service-oriented/pouring-cold-water-on-soa-reuse-mantra/699 adresinden alındı

Microsoft. (2008). *Code Metrics Overview*. 4 20, 2010 tarihinde Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/bb385914.aspx adresinden alındı

Microsoft. (2010, 3 21). *Microsoft Visual Studio Team System*. 4 19, 2010 tarihinde Microsoft Developer Network: http://msdn.microsoft.com/en-us/vstudio/default.aspx adresinden alındı

Mikovec, Z., Vystrcil, J., & Slavik, P. (2009, Haziran). Web toolkits accessibility study. *ACM SIGACCESS Accessibility and Computing*(94), s. 3-8.

Mueller, J. (2007). *Microsoft IIS 7 Implementation and Administration.* Alameda, CA, USA: SYBEX Inc.

Nagel, W. (2005). *Subversion Version Control: Using the Subversion Version Control System in Development Projects.* NJ, USA: Prentice Hall PTR.

OpenNLP. (2010). 5 1, 2010 tarihinde OpenNLP Project: http://opennlp.sourceforge.net/ adresinden alındı

Parasoft. (2009). *Online Documents*. 3 14, 2010 tarihinde Jtest Documentation: http://www.parasoft.com/ adresinden alındı

Pathan, M., & Buyya, R. (2009). Architecture and performance models for QoS-driven effective peering of content delivery networks. *Multiagent and Grid Systems*.

Ravi, J., Yu, Z., & Shi, W. (2009). A survey on dynamic Web content generation and delivery techniques. *Journal of Network and Computer Applications, 32*(Issue 5).

Ricardo A. Baeza-Yates, B. R.-N. (1999). *Modern Information Retrieval.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Souders, S. (2008). High Performance Web Sites. *6*(Issue 6).

Strzalkowski, T. (1999). *Natural Language Information Retrieval.* Norwell, MA, USA: Kluwer Academic Publishers.

Stuckenholz, A. (2005). Component evolution and versioning state of the art. *ACM SIGSOFT Software Engineering Notes*, 7.

Tan, S.-W., Waters, G., & Crawford, J. (2003). *A Survey and Performance Evaluation of Scalable Tree-based Application Layer Multicast Protocols.* University of Kent, UK: Computing Laboratory.

The Dojo Foundation. (2010). *Dojo web toolkit*. Mayıs 15, 2010 tarihinde http://www.dojotoolkit.org/ adresinden alındı

The Unicode Consortium. (2003). *The Unicode Standard, Version 4.0.* Boston, MA, USA: Addison-Wesley Longman Publishing.

Tillmann, N., & Schulte, W. (2005). Parameterized Unit Tests. *Foundations of Software Engineering*, 253-262.

Trieu, C., Ajay, M., Alexei, A., & Alla, S. (2009). Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. *IEEE International Conference on e-Business Engineering*, 281-286.

W3C. (2008). *Web Accessibility Initiative*. Mayıs 25, 2010 tarihinde http://www.w3.org/WAI/ adresinden alındı

W3C. (2009). *SOAP Tanımları Sürüm 1.2*. Mayıs 2010 tarihinde http://www.w3.org/TR/soap/ adresinden alındı

W3C. (2010). *Accessible Rich Internet Applications Suite*. Mayıs 25, 2010 tarihinde http://www.w3.org/WAI/intro/aria adresinden alındı

Webservices.org. (2005, November). *Study Confirms Service Reuse Reduces Project Cycle Time.* May 2010 tarihinde http://www.webservices.org/categories/management/study_confirms_service_reuse_reduces_project_cycle_time adresinden alındı

Wheeler, D. (2005). Comments on Open Source Software / Free Software (OSS/FS) Software Configuration Management (SCM) / Revision-Control Systems.

Whitehead, E. (2001). Design spaces for link and structure versioning. *Proceedings of the 12th ACM conference on Hypertext and Hypermedia*, 195-204.

Woo, T., & Lam, S. (1993). *Designing a Distributed Authorization Service.* Austin, TX, USA: University of Texas.

WWWC. (2010, 3 4). *HTML 5 Specification*. 4 1, 2010 tarihinde World Wide Web Consortium: http://www.w3.org/TR/html5/ adresinden alındı

Yang, L., Zhang, Y., & Li, H. (2009). Information Search Based on Test Mining and EXTJS. *Proceedings of the 2009 WRI World Congress on Software Engineering*, s. 386-391.

Yao, D., & Tamassia, R. (2009). Compact and Anonymous Role-Based Authorization Chain. *ACM Transactions on Information and System Security (TISSEC)*, No.: 15.

Yergeau, F. (2003). *RFC3629: UTF-8, a transformation format of ISO 10646.* USA: RFC Editor.

Zhang, C., & Cordes, D. (2004). Simulation of resource synchronization in a dynamic real-time distributed computing environment: Research Articles. *Concurrency and Computation: Practice & Experience*, 1433-1451.

EK 1 – Kaynak Kodları

Açık kaynak kodlu bir proje olan GOTA platformunun kaynak kodlarına http://gota.codeplex.com adresinden ulaşılabilir.

EK 2 – Uygulama Arayüzü

GOTA platformu http://www.gota-project.org adresinde araştırmacıların kullanımı için yayınlanmış durumdadır.