WEB TABANLI DUBLIN CORE METADATA ÜRETİCİSİ TASARIMI

WEB BASED DUBLIN CORE METADATA GENERATOR DESIGN

SERMET SOYKAN

Başkent Üniversitesi
Lisansüstü Eğitim Öğretim ve Sınav Yönetmeliğinin
İstatistik ve Bilgisayar Bilimleri Anabilim Dalı İçin Öngördüğü
YÜKSEK LİSANS TEZİ
olarak hazırlanmıştır.

2007

Fen Bilimleri Enstitüsü Müdürlüğü'ne,

Bu çalışma, jürimiz tarafından **İSTATİSTİK VE BİLGİSAYAR BİLİMLERİ ANABİLİM DALI 'nda YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Başkan	:	Prof. Dr. İsmail ERDEM
Üye (Danışman)	:	Prof. Dr. Timur KARAÇAY
Üye	:	Yard. Doç Dr. Harun ARTUNER

ONAY

Bu tez 28/09/2007 tarihinde Enstitü Yönetim Kurulunca belirlenen yukarıdaki jüri üyeleri tarafından kabul edilmiştir.

...../...../.....

Prof.Dr. Emin AKATA

FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRÜ

TEŞEKKÜR

Tezimin hazırlanması esnasında her türlü yardımını esirgemeyen ve büyük destek olan hocam, Sayın Prof. Dr. Timur KARAÇAY'a (tez danışmanı), tez çalışmalarım esnasında yardımını gördüğüm hocam Sayın Yard. Doç. Dr. Harun ARTUNER'e ve çalışmalarım esnasında bana destek olan Aileme teşekkür ederim.

ÖZ

WEB TABANLI DUBLIN CORE METADATA ÜRETİCİSİ TASARIMI

Sermet SOYKAN

Başkent Üniversitesi Fen Bilimleri Enstitüsü

İstatistik ve Bilgisayar Bilimleri Anabilim Dalı

Internetin sağladığı erişim kolaylığı web ortamına giren bilgi miktarını her geçen gün büyük ölçüde artırmaktadır. Öte yandan, durmadan artan bu büyük bilgi ambarından istenilen bilginin seçilmesi giderek zorlaşmaktadır. Bu nedenle, web dökümanlarında bilgi hakkında bilgi veren METADATA kullanımının standartlaştırılması için yapılan çalışmalar önem kazanmıştır. Bunun sonucu olarak, metadata yardımı ile istenilen bilgiye erişimi sağlayacak programların ortaya çıkması doğaldır.

Bu çalışmada adresi verilen bir web dökümanı hakkında bilgi oluşturacak "Web Tabanlı Bir Otomatik Metadata Üretme" programı tasarlanıp gerçekleştirilmiştir. Tasarlanan program Web ortamındaki sayfanın kaynak kodunda bulunan "META" takılarının yanı sıra sayfada bulunan iç ve dış bağlantıları da listelemekte ayrıca belirtilen sayfada tekrarlanan veya koyu yazılarak vurgulanmış sözcükleri kullanıcıya göstermektedir. Elde edilen "META" takılarının içerikleri ile kullanıcının girdiği verileri kullanarak, tercihe göre, RDF/XML, Dublin Core/XML ve HTML dökümanları için Dublin Core türünde metadata oluşturabilmektedir. Microsoft Visual Studio.Net ortamında C# ve ASP dilleriyle geliştirilen ve Türkçe karakterleri destekleme yeteneğine sahip kılınan program "html" uzantılı sayfalarda başarıyla denenmiştir.

ANAHTAR SÖZCÜKLER: Metadata, metadata oluşturma, Dublin Core, RDF

Danışman: Prof. Dr. Timur KARAÇAY, Başkent Üniversitesi, İstatistik ve Bilgisayar Bilimleri Bölümü.

ABSTRACT

WEB BASED DUBLIN CORE METADATA GENERATOR DESIGN

Sermet SOYKAN

Başkent University Graduate School of Natural and Applied Sciences

Statistics and Computer Science

The freedom of dissemination and the attainability of knowledge on the internet cause the accumulation of web documents significantly day by day. Therefore, it is getting more and more difficult to select the appropriate documents for the knowledge one may request from this huge document base. It is this reason that a standardization is needed for the usage of METADATA, which provides information about web documents. Usage of metadata gained a great importance nowadays. It is then natural to expect a great accumulation of the quantity of programs or tools facilitating the attainment proper knowledge on the web via metadata.

In this thesis, we designed and developed a “Web based automatic metadata generator“ program retrieving information from a given Web page. The program not only parses “META” tags but also has the quality of being able to list internal and external links as well as the repeated and bold written words on the page. The contents of the tags obtained may be mixed with the user’s data to generate metadata tags of Dublin Core type for RDF/XML, XML and HTML documents, according to the preferences of the user. The program has been developed by the use of Microsoft Visual Studio.Net 2005 environment, ASP and C# languages, and has been tested on web pages with html extensions successfully.

KEYWORDS: Metadata, metadata generator, Dublin Core, RDF

Advisor: Prof. Dr. Timur KARAÇAY, Başkent University, Statistics and Computer Science Department.

İÇİNDEKİLER LİSTESİ

ŞEKİLLER LİSTESİ

Sayfa

TABLOLAR LİSTESİ

Sayfa

KISALTMALAR LİSTESİ

CSDGM	Content Standart for Digital Geospatial Metadata
CSS	Cascading Style Sheets
DC	Dublin Core
DCMES	Dublin Core Metadata Element Set
DCMI	Dublin Core Metadata Initiative
EAD	Encoded Archival Description
FGDC	Federal Geographical Data Committee
GILS	Government Information Locator Service
GML	Generalized Markup Language
HTML	Hypertext Markup Language
ISO	International Organization for Standardization
MARC	MAchine Readable Cataloging
OAI	Open Archives Initiative
ONIX	Online Information Exchange
PDF	Portable Document Format
PICS	Platform for Internet Content Selection
PICS NG	Platform for Internet Content Selection Next Generation
RDF	Resource Description Framework
SGML	Standart Generalized Markup Language
TEI	Text Encoded Initiative
VRA	Visual Resources Association
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1. GİRİŞ

World Wide Web'in ortaya çıkması bilgiye erişimde çok büyük kolaylıklar sağlamıştır. Daha önceleri bir kavramı araştırmak günler hatta haftalar alırken artık internet bağlantısı olan bir kullanıcı istediği bilgiye birkaç tuşa dokunarak kolaylıkla ulaşabilmektedir. İnternet'in ortaya çıkma amacının bilgi değiş tokuşunu kolaylaştırmak olduğu düşünülürse bilgiye ulaşmanın kolaylaşması internetin beklenen bir sonucu olmuştur. İnternet'in hızlı bir biçimde gelişmesi ve hayatımızla bütünleşmesi, internet üzerinden erişilebilen bilginin tür ve miktarını büyük ölçüde arttırmıştır. Şu an internet üzerinde yemek tarifinden çocuk bakımına, bilgisayardan balıkçılığa kadar pek çok konuda bilgi bulunmaktadır. 2003 yılında dünya çapında kişi başına 800 MB veri üretildiği belirtilmektedir [18]. Dünya çapında ise her yıl iki exabyte[1] kadar veri üretildiği tahmin edilmektedir [17]. Halen üretilen kaynakların %90'ı ya dijital ortamda üretilmekte ya da kâğıt üzerine basıldıktan sonra dijital ortama aktarılmaktadır. Dijital ortama aktarılan veri sayısı gün geçtikçe artmaya devam etmektedir. İnternet arama motorlarından Google, birkaç yıl içinde yaklaşık 15 milyon kitabı dijital ortama aktarmak üzere bir kütüphane çalışması yaptığını açıklamıştır. Yapılan hesaplamalara göre bir yıl içerisinde üretilen basılı materyaller elektronik ortama aktarıldıklarında 50TB ile 200 TB arasında yer kapladıkları belirtilmektedir [18].

İnternet üzerinde bulunan bilgi miktarının artması bu bilgiye erişim sorununu da beraberinde getirmektedir. Kullanıcıların istedikleri bilgileri bulmalarını kolaylaştırmak adına ortaya çıkan arama motorları bugün internetin vazgeçilmezleri arasındadır. Birkaç anahtar kelime girmek suretiyle istenilen Web sitesine ya da bilgiye erişmek hiç de zor değildir. Kullanıcıların bilgiye erişimini kolaylaştıran arama motorları Web sayfalarını indekslemek için sayfanın içeriğini kullanmanın yanında bütün internet kullanıcılarının duyduğu ancak fazla bilgi sahibi olmadıkları metadataları da kullanmaktadır. Web üzerinde aranılan bilgiye

[1] Bir exabyte bir milyar gigabyte'a, yani 10^{18} byte'a eşittir. Bilgisayar ortamında bir harf, rakam ya da simge depolamak için bir "byte"lık kapasite gerekmektedir. Kabaca bin byte, bir kilobayt'a (KB), bir milyon byte bir megabyte'a (MB), bir milyar byte bir gigabyte'a (GB), bir trilyon byte bir terabyte'a (TB), bir katrilyon byte bir exabyte'a (EB) eşittir.

erişim için metadata bu sebeple çok önemli olmasına rağmen son kullanıcılar ve kaynak oluşturan kişiler tarafından yeterince önemsenmemektedir. Kendi hazırladıkları Web sayfalarının erişim sayısını arttırmak isteyen amatör veya kötü niyetli Web tasarımcıları dışında metadatanın önemini doğru algılayan insan sayısı fazla değildir. Web üzerine eklenen bilginin her geçen saniye arttığı düşünülürse bilginin indekslenmesi konusunda kapsamlı çalışmalar yapılmazsa yakın bir dönemde internet üzerinde aranılan bilgiye erişmek oldukça zorlaşacaktır.

Bilgiye erişimdeki problemler sadece aranılan bilgi için arama motoruna birkaç anahtar kelime yazmakla çözülmemektedir. Erişilen bilginin doğru ve güvenilir bilgi olması da son derece önemlidir. Tarihin eski zamanlarından bu yana bilgiye erişimdeki zorluklar düşünürlerin dikkatini çekmiştir. Söz gelimi Plato'nun, "Meno İkilemi"[2] doğru bilgiyi arama ve bulmaktaki güçlükleri ortaya koymaktadır. Bu nedenle Web gibi bilgi yüklemesinin (*Information Overload*) yoğun olduğu bir ortamda istenilen ve doğru bilgiye erişimi kolaylaştırmak için son dönemlerde metadata üretme ve standartlaştırma çalışmaları oldukça önem kazanmıştır.

Web üzerinde aranılan bilgiyi bulmayı kolaylaştıracak teknolojilerin geliştirilmesi bütün bu anlatılan gelişmelere paralel olarak artmıştır.

1.1 Önbilgi

1.1.1 İşaretleme dili tanımı

Bütün dokümanlar içerik ve bu içeriği görsel ya da anlamsal olarak şekillendiren işaretlemelerden ibarettir. Bir dokümanın içeriğini harfler, şekiller ve ona içeriğini kazandıran diğer kısımlar oluşturur. Ancak bu öğelerin tek başlarına bir araya gelmesi okunabilir bir doküman oluşturmak için yeterli gelmemektedir. Bu nedenle içeriğe biçim verecek işaretleme dillerine ihtiyaç duyulmaktadır. Burada

[2] Meno, Socrates'e "Doğruyu neden arıyoruz? Eğer neyin doğru olduğunu biliyorsak aramak gereksizdir. Eğer neyin doğru olduğunu bilmiyorsak bulduğumuzu nasıl anlarız?" sorusunu sormuştur.

işaretlenenler kullandığımız veriler yani dokümanımızın içeriğidir. İşaretleme dilleri bizlere dokümanların yapısı ve biçimi hakkında bilgi vermektedir. Yapısal ve Biçimsel olmak üzere iki çeşit işaretleme vardır. Yapısal işaretleme (Tanımsal) ile dokümanımızın mantıksal olarak nasıl bölümlendiği (paragraf, kısım vb.) ve bu parçaların hiyerarşik olarak nasıl organize edildiği belirtilir. Biçimsel işaretleme (Prosedürel/Sunumsal) ise günlük hayatımızda kelime işleme programları ile sıkça yaptığımız gibi yazı tipi, puntosu, satır aralığı gibi dokümanın sunumsal yapısı ile ilgili işlemler yapılmasını sağlamaktadır. Günlük hayatımızda genel olarak yarattığımız dokümanların görünüşü ile ilgilendiğimiz için biçimsel işaretleme daha sık kullanılmaktadır. Yukarıda sözü edilen bu iki işaretleme birbirlerinden bağımsız gibi görünseler de gerçekte durum böyle değildir. Sözgelimi bir Word dokümanında bir metni 12 punto, Ariel ile sola dayalı olarak sadece biçimsel işaretleme ile belirleyebilmek mümkündür ve sunumsal açıdan istenilen yapılmış olabilir. Ancak bu işaretleme ile seçilen bu metnin bir başlık bilgisi mi yoksa sadece yazıda geçen bir pasaj mı olduğu hakkında bir bilgiye sahip olunamaz. Bu durum bizler için bir problem gibi gözükmese de, bu kısımlar yapısal olarak işaretlenmediği için Word programı ile bu başlık bilgilerini kullanarak otomatik olarak içindekiler oluşturmak mümkün olmayacaktır. Bunun yanında eğer ilerde sadece başlıkların görünüş biçimi değiştirmek istenirse de bunu otomatik olarak yapma olanağı kalmayacaktır. Bunun yerine eğer başlıkların biçimsel özellikleri (yazı puntosu, türü vb.) tanımlanırsa yapısal işaretleme yapıldığında aynı zamanda otomatik olarak biçimsel işaretleme de yapılmış olur. Böylece başlık bilgilerinde bir değişiklik yapıldığında bunun biçimsel olarak da yapılması mümkün olacaktır. Bu şekilde dokümanların belirli bazı takılarla işaretlenmesi dokümanın ileriki zamanlarda özellikle insan müdahalesi olmadan bilgisayarlar tarafından otomatik işlenmesi açısından oldukça kolaylık sağlamaktadır.

Dokümanların görüntüsünden ziyade yapısı ile ilgilenen bir işaretleme diline olan ihtiyaç SGML gibi işaretleme dillerinin doğmasına sebep olmuştur. İşaretleme dilleri oluşturulurken belirli bazı gereksinimleri sağlamaları istenmektedir. İşaretleme dilleriyle tanımlayıcı takılar arasında hiyerarşik bir düzen oluşturulup dokümanların belirli kısımlarının tanımlanması ve düzene konmasının sağlanması gerekmektedir. İşaretleme dilinin genişletilebilir olması yani amaca göre istenilen

yeni takıların eklenebilmesi bir diğer gereksinimdir. Son olarak da oluşturulan bu işaretleme dilinin bir sahibinin olmaması isteyen herkesin özgürce kullanabilmesi gerekmektedir. Yukarıda sayılan özellikler SGML tarafından sağlanmaktadır.

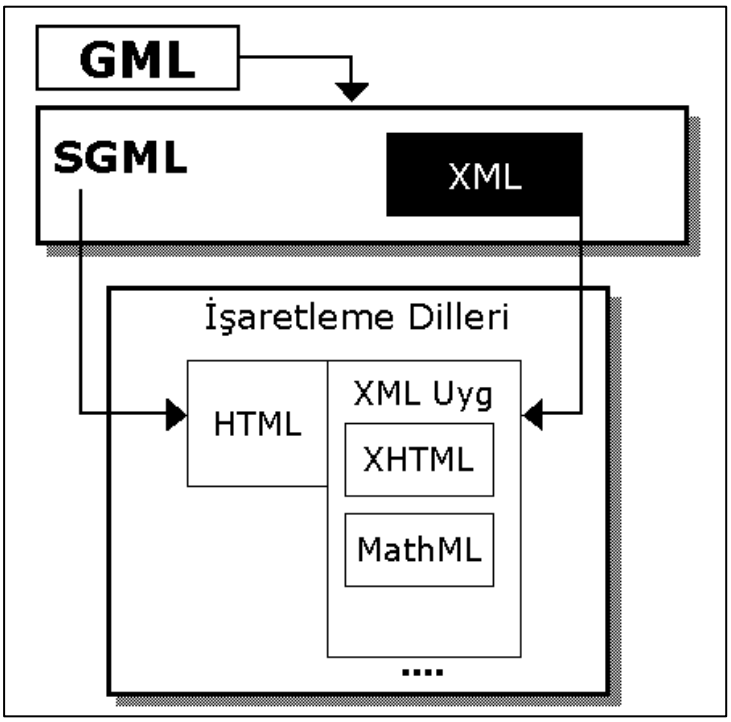
1.1.2 İşaretleme dillerinin gelişimi

Kaynakların el ile yazıldığı dönemlerde yayıncılar dokümanların kenarlarına yazılacak kısmın stil ve boyut gibi bilgilerini yazmaktaydılar. Böylece dokümanları yazma işi ile uğraşan kişi kolaylıkla hangi kısmı hangi stil ile yazacağını bilebilmekteydi. Metinin belirli kısımlarının özel anlam kazandırmak amacıyla işaretlenmesi işlemine işaretleme (*Marking Up*) denilmektedir. Bu kavram el yazmalarından çok sonra bilgisayar terimleri arasına da girmiş ve elektronik ortamdaki dokümanların görüntülenme özellikleri ile ilgili olarak kullanılmaktadır.

1.1.2.1 SGML (Standard Generalized Markup Language)

Standart Generalized Markup Language'in kısaltması olan SGML, HTML, XML gibi işaretleme dilleri (*Mark-Up Language*) yaratmaya yarayan bir meta dilidir. SGML, platform ve sistemden bağımsız olarak kullanıcıların istedikleri dokümanlara erişmesine izin veren bir dildir. İnternet ile birlikte Web üzerinde yayınlanan bir yazı çok farklı bilgisayar platformlarında farklı özelliklere sahip bilgisayarlar tarafından görüntülenmeye başlamıştır. Yazılı dokümanların kâğıt üzerinden bilgisayarlara geçme sürecinin başlaması ile beraber bu dokümanları yapılandıracak bir dile olan ihtiyaç ortaya çıkmıştır. Metin ve belgelerin kolay bir şekilde taşınabilmesi, paylaşılabilmesi ve işlenebilmesi için ilk işaretleme dili GML (*Generalized Markup Language*), 1960 sonlarında IBM'de yapılan araştırma çalışmaları sonunda ortaya çıkmıştır. GML, IBM bünyesinde geniş çaplı olarak kullanılmıştır. 1978 yılında Amerikan Ulusal Standartlar Enstitüsü (ANSI), bilgi değişimini ülke çapında standart hale getirmek için bir işaretleme dili üzerinde çalışmaya başlamıştır. 1980 yılında çalışmaların ilk taslağı yayınlanmış 1983 yılında da standart hale getirilmiştir. 1984 yılında bu standartlaştırma çalışmalarına Uluslararası Standartlar Teşkilatı (ISO) da katılmış ve 1986 yılında SGML yapısı

ortaya çıkmıştır. SGML, özellikle yayıncılıkta kullanılmış ancak çok esnek bir yapıya sahip olması ve yüksek uygulama geliştirme maliyetinden dolayı Web üzerinde geniş çaplı kullanılamamıştır. SGML, dokümanların yapısını tanımlayan güçlü bir dil olmasının yanında yeni diller yaratmaya izin veren bir gramer oluşturma yeteneğine de sahiptir. SGML, Web üzerinde fazla kullanılmasa da ondan türetilen diller (HTML, XML vb.) büyük ölçüde kullanılmaktadır. SGML, kullanıcılara temel bir sözdizimi sağlayarak kullanıcıların kendi elemanlarını tanımlamasına izin vermektedir. SGML ile tüm platformlarda aynı şekilde görüntülenebilecek dokümanlar yaratmak hedeflenmiştir. SGML yapısına göre bir doküman genel anlamda iç içe geçmiş elementlerin (Bölüm, kısım, paragraf vb.) hiyerarşik bir yapısıdır. SGML'in en önemli özelliği dokümanların görsel yapısı ile ilgilenmemesidir. Ancak stil sayfaları ya da prosedürel işaretleme dilleriyle ilişkilendirilerek SGML'e görsellik kazandırmak mümkündür. SGML dokümanlarını metin editörleri ile hazırlamak mümkün olduğu gibi bunun için tasarlanmış programlarla da oluşturmak mümkündür. SGML ile birlikte *Document Type Definition* (DTD) Belge Tür Tanımlaması kavramını da açıklamak gerekmektedir. Yukarıdaki bölümde bir işaretleme dilinin genişletilebilir yani amaca yönelik olarak yeni takılar tanımlanabilir olması kavramından bahsedilmişti. Bu şekilde takılar oluşturmaktaki en büyük sorun farklı bilgisayarların oluşturulan bu takıları anlamasını sağlamaktır. Yani oluşturulan bir takı oluşturulduğu bilgisayara anlamlı gelirken başka bir bilgisayarda bir anlam ifade etmeyebilir. Oluşturulan bu takıların birbirleri ile olan ilişkilerindeki kurallar DTD ile belirlenmektedir. DTD' yi bir anlamda oluşturulan takıların kullanılma kurallarını belirleyen bir şablon olarak görmek mümkündür. SGML'de HTML'in aksine DTD' yi kullanıcıların kendisinin oluşturması mümkündür. DTD' yi oluşturulan dokümanın içine gömmek mümkün olduğu gibi ayrı bir dosya olarak referans gösterebilmek de mümkündür. SGML'in daha önce belirtildiği gibi karmaşık bir yapıya sahip olması farklı ve basitleştirilmiş dillerin ortaya çıkmasına sebep olmuştur.



Şekil 1.1: İşaretleme Dillerine Genel Bakış

1.1.2.2 HTML (HyperText Markup Language)

HTML, tarayıcılardan görülebilecek (Internet Explorer, Netscape vb.) internet dokümanlarını yaratmaya yarayan bir işaretleme dilidir. 1990 yılında Tim Berners Lee tarafından CERN'de (Avrupa Nükleer Araştırma Laboratuvarı) bilim adamlarının dokümanlarını meslektaşları ile paylaşabilmelerini sağlamak maksadıyla geliştirilmiş ve daha sonra tüm dünyaya yayılmıştır. SGML'in geliştirilmesi ile dokümanların farklı bilgisayar ve platformlarda görüntülenme sorunu çözülmüş gibi görünse de SGML'de bulunan bazı eksiklikler yeni dillerin geliştirilmesi gerekliliğini ortaya koymuştur. SGML'in esnek kuralları SGML ile oluşturulan dokümanların tarayıcılar tarafından işlenebilmesini oldukça zorlaştırmaktadır. Bunun yanında SGML bir meta dili olduğu için dokümanların görüntülenmesi ile ilgilenmemektedir. HTML, SGML'in bir meta dili olması özelliği sayesinde türetilmiş, onun takılarının belirli bir bölümünü kullanan basitleştirilmiş bir SGML sözlüğüdür. HTML'in asıl amacı dokümanları Web üzerinde bulunan farklı bilgisayarlarda aynı şekilde görüntülemektir. Yani HTML, dokümanların

anlamsal yapısı ile değil sadece gösterimi ile ilgilidir. HTML'de sabit takılar bulunmaktadır ve kullanıcının kendi takılarını tanımlama (genişletilebilme) imkânı bulunmamaktadır. HTML, takılar konusunda katı bir dil değildir. Açılan takılar kapanmaz ya da birbirleri içine uygun biçimde yuvalanmazlarsa da tarayıcılar HTML dokümanlarını görüntüleyebilmektedirler. HTML'in, SGML'den bir diğer farkı da sabit bir DTD' ye sahip olmasıdır. Tarayıcılar bu sabit DTD' ye göre takıları yorumlamaktadırlar. HTML'in, 1989'dan bu yana 4 sürümü çıkmıştır. Halen 4.01 sürümü kullanılmaktadır. HTML 5 için W3C tarafından çalışmalar devam etmektedir.

1.1.2.3 XML (Extensible Markup Language)

Daha önce SGML'in dokümanların görüntülenmesinden ziyade anlamsal özellikleri ile ilgilendiği belirtilmişti. Ancak SGML'in tarayıcılarda görüntülenmesinin zor olması basitleştirilmiş SGML olan HTML'in doğmasına neden oldu ve Web üzerinde doküman görüntülemek için HTML kullanılmaya başlandı. Ancak HTML'in dokümanların anlamsal özelliklerini belirtme özelliğinin olmaması bilişimcileri yeni diller üzerinde çalışmaya yöneltti. 1996 yılında W3C XML'i (*Extensible Markup Language*) geliştirdi. XML, dokümanların yapısını ve anlamsal özelliklerini yansıtabilen bir dildir. HTML dokümanların sadece ekran üzerinde gösterimi ile ilgilenirken XML, SGML' deki anlamsallığı tekrar devreye sokmuştur. XML sadece bir işaretleme dili değildir. HTML, SGML ile yazılmış bir işaretleme dili iken XML, SGML'in alt kümesi olan bir meta dilidir. Bu nedenle, XML ile kullanıcılar kendi işaretleme dillerini oluşturabilmektedir. Birçok kaynakta XML, SGML'in daha az karmaşık ama aynı güce sahip hali olarak gösterilmektedir. XML, SGML'in olumlu yönlerini alıp karmaşık bazı yönlerini atmıştır. XML de sabit olmayan bir DTD yapısına sahiptir. Kullanıcı tanımladığı takılara göre DTD oluşturabilmektedir. Bu DTD aynı zamanda takıların nasıl kullanılacağı konusunda bilgi vermektedir. XML, HTML gibi esnek kurallara sahip değildir. Açılan takıların mutlaka kapanması, takıların uygun biçimde yuvalanması ve takılar yazılırken büyük-küçük harfe dikkat edilmesi gibi katı kuralları bulunmaktadır. Eğer bu kurallara dikkat edilmezse XML dokümanı iyi biçimlenmiş (*Well-Formed*) olmaz. Bunun yanında XML dokümanının oluşturulan DTD'ye uygun olması onun geçerli (*Valid*) olmasını sağlar. XML takıları

kullanıcılar tarafından tanımlandığı için tarayıcılar XML dokümanlarını sadece kaynak kod şeklinde gösterebilmektedirler. XML dokümanlarına görsellik kazandırmak isteniyorsa o XML dokümanına CSS (*Cascading Style Sheet*) ya da XSL (*Extensible Stylesheet Language*) gibi bir stil dosyası bağlamak gerekmektedir.

1.1.2.4 XHTML (Extensible HyperText Markup Language)

Önceki bölümlerde HTML'in kurallarının katı olmadığından bahsedilmişti. Yani açılan bir takının kapanmaması ya da büyük-küçük harfe dikkat edilmemesi gibi özellikler HTML dokümanının tarayıcılar tarafından görüntülenmesinde herhangi bir sorun çıkarmamaktadır. İnternet'in kullanılmaya başlandığı ilk zamanlarda sadece masa üstü bilgisayarlar kullanılmaktaydı. Bu sebeple yukarıda sayılan HTML hataları bilgisayarlar üzerinde çalışan tarayıcılar tarafından yorumlanıp yok sayılabiliniyordu. Ancak teknolojinin ilerlemesi ile beraber cep telefonları ve avuç içi bilgisayarlardan da internet erişimi mümkün oldu. Bu cihazların bu gibi hataları yorumlayıp yok saymaya yeterli kaynakları bulunmamaktaydı. Bu sebeple HTML'in daha temiz ve kurallar açısından katı bir sürümüne ihtiyaç duyuldu. W3C 26 Ocak 2000 tarihinde XHTML 1.0 tavsiyesini yayınladı. XHTML, HTML'in bütün takılarının XML kuralları ile yazılmış halidir. Başka bir deyişle XHTML, XML sözdiziminin HTML içinde kullanılmış halidir. Bu dil ile XML'deki gibi iyi yapılanmış dokümanlar yaratıp bu dokümanları bilgisayar ve mobil cihazlar üzerindeki tarayıcılarda kullanmak mümkün olmaktadır.

1.2 Web Üzerinde Bilgiye Erişim Teknikleri

World Wide Web, veriye kolay ulaşma konusunda devrim yaratıp kullanıcılara büyük kolaylıklar sağlamış olsa da, güvenilir ve etkili bir küresel bilgi erişim aracı olmak adına yapısında bazı eksiklikleri barındırmaktadır.

Kullanıcılar, Web üzerinden birçok farklı alandaki veriye erişme imkânına sahip olsalar da Web üzerine aralıksız bilgi eklenmesi bilgiye erişimde bazı güçlükler

yaşatmaktadır. Bilgiye erişim sorunu gün geçtikçe büyümeye devam etmektedir. Web'in dünya üzerindeki en büyük bilgi edinme kaynağı olmasını sağlayan evrensellik aynı zamanda Web üzerinde aranılan bilgiyi bulmayı zorlaştıran bir faktör olarak kullanıcıların karşısına çıkmaktadır. Web üzerinde kullanıcıların erişimine açık olan bilgi farklı yapı, format ve içeriktedir. Ancak Web, bu bilgilere kolay erişime imkân verecek birörnek (tekbiçimli) bir plana sahip değildir. Web üzerinde bulunan bilgiye erişmekteki zorlukların başlıca sebepleri şunlardır:

1. Web üzerinde bulunan bilgilerin yapılandırılmamış olması:

HTML'in en zayıf tarafı verinin kullanıcı ekranında görüntülenmesi için bir yapı sunmasına rağmen veri hakkında herhangi bir bilgi içermemesidir. HTML'de kullanılan takılar bilgisayarlara sadece sayfa üzerine yerleştirilme ve görüntü hakkında bilgi vermektedirler. Bu sebeple bilgisayarlar yazılan metinlerin ekrana gelme şekli ile ilgili bilgi sahibi olurken bilginin içeriği ile ilgili bir bilgiye sahip olmazlar. Verileri yapılandırılmış (*Structured*) ve yapılandırılmamış (*Unstructured*) olmak üzere ikiye ayırabilmek mümkündür. Yapılandırılmış veri ile XML belgesi, veri tabanı ve katalog dokümanları belirtilirken yapılandırılmamış veri ile e-posta ve PDF dokümanları belirtilmektedir. Her ne kadar HTML verilerin sayfa üzerinde gösterimini belirli bir yapıya soksa da bu sadece gösterim ile ilgili bir yapılandırmadır. Bu sebeple bilgisayarların bu verileri anlaması mümkün değildir. İleriki bölümlerde açıklanacak olan Metadata'ların Web sayfalarının içeriği hakkında bilgi vermesi sayesinde bilgisayarların görüntüledikleri veriler hakkında bilgi sahibi olup onları kullanmaları sağlanmış olur. Ancak buradaki veriyi kullanma kavramı bilgisayarların yazılan verileri öğrenecekleri bir yapay zekâ sistemini değil, bilgisayarların gösterdikleri verilerin içeriği hakkında bilgi sahibi olmaları anlamına gelmektedir. Web üzerine eklenecek kaynakların tümüne metadata eklenmesi ile veri erişim araçlarının aranılan bilgiye erişmesinde büyük kolaylıklar sağlanmış olur.

2. Dinamik ve dağıtık bir ortam olması:

Web üzerinde bulunan bilgiler dağıtık bir ortamdadır. Yani bilgi parçaları farklı şehir ve hatta ülkelerde bulunan sunucu bilgisayarlar üzerinde bulunabilir ve bu bilgilerin anlamlı bir hale getirilebilmesi için bunların bir araya getirilmeleri gerekebilir. Örneğin bir sitede İzmir'de saat kulesi vardır bilgisi yer alırken başka bir sitede İzmir'deki festivaller bilgisi varsa bu iki bilgiyi birleştirip saat kulesi olan şehirlerdeki festivaller sorgusunu cevaplayacak bir yapının bu iki sitede bulunan bilgilerden yararlanması gerekmektedir. İnternet'in bir sonraki aşaması olarak düşünülen "Anlamsal Web" ile Web'in veri tabanlarının oluşturduğu bir ağ yapısı haline geleceği düşünüldüğünde bu yapının oluşturulabilmesi için bilgisayarların farklı yerdeki verilerin içeriğini anlaması gerekmektedir. Bunun sağlanabilmesi için de bilgisayarların sahibi oldukları verinin içeriği hakkında bilgi sahibi olmaları gerekmektedir.

Elektronik ortamdaki veriye erişimdeki bir diğer sorun, erişimin kullanılan donanımlara bağlı olmasıdır. Bazı sitelere teknik sebeplerden ötürü (değişen URL veya sunucu arızası gibi) ulaşmak mümkün olamayabilmektedir. Özellikle Web üzerindeki kaynakların adreslerinin değişimi aranılan bilgiye erişimi oldukça zorlaştırmaktadır. Alexa'nın bir araştırmasına göre Web kaynakları ortalama olarak 75 günde kaybolmaktadır [7].

3. Çok hızlı bir şekilde büyümesi:

Web üzerine her geçen saniye yeni veriler eklenmekte ve eklenen bu verilerin çoğunluğunu tekrar ya da işe yaramayan veriler oluşturmaktadır. İnternet kullanıcıları, gereksinim duymadıkları, özümseyemedikleri ya da işlem yapma zamanı bulamadıkları birçok veriyle karşılaşmaktadır. İngiliz Haber Ajansı Reuters'de saniyede 27,000 sayfa belge üretilmektedir [17]. Bu kadar çok bilgiyi insan gücü ile işlemek mümkün değildir.

Web üzerindeki erişim sorununu çözmek için verileri indeksleyen arama motorları ortaya çıkmıştır. Web üzerindeki sürekli artan bilgiyi sınıflandırmak ve belirli bir kullanıcı için aradığı konu ile ilgili veriyi sunmak önemli bir görev halini almıştır.

Arama motorlarının ortaya çıkması veriye erişimi büyük ölçüde kolaylaştırmış olsa da bilgiye erişim sorunu tamamıyla çözülmüş değildir. Arama motorlarının istediğimiz bilgileri bize verirken karşılaştığı bazı zorluklar vardır. Bu zorluklar:

- **Web'in Hızlı Büyümesi**

Web, şu anki arama motorlarının indeksleme hızından çok daha hızlı bir şekilde büyümektedir. Ortalama olarak her sekiz ayda bir Web içeriği ikiye katlanmaktadır. Bu hızlı artış arama motorlarının ulaşamadığı veri sayısını da arttırmaktadır.

- **Web Sitelerinin Sık Güncellenmesi**

Çoğu Web sitesi sıkça güncellenmekte yani içeriği değişmektedir. Web'in Akışkan doğası (*Fluid Nature*) yani kaynakların sürekli değişimi bir kez indekslemeyi yeterli kılmamaktadır. Geleneksel bilgi erişim sistemlerinde (kütüphane vb.) bir kaynak sadece bir kez indekslenirken Web üzerinde bulunan kaynakların içeriği sürekli değiştiğinden yapılan indekslemenin belirli aralıklarla tekrarlanması gerekmektedir.

- **İlgisiz Metadata Kullanımı**

Bazı Web siteleri arama motorlarında üst sıraya yükselebilmek ve sitelerinin ziyaretçi sayılarını arttırabilmek için ilgisiz metadata'lar kullanmakta ve arama motoru ve kullanıcıyı yanıltmaktadırlar.

- **Aranılan Kelimenin Birden Fazla Disiplinde Geçmesi**

Aranılan anahtar kelime birden fazla disiplinde geçiyorsa arama motorunun döndürdüğü sayfa sayısı milyonları bulmakta (*False Positive Problem*) ve kullanıcının istediği bilgiye erişmesi zorlaşmaktadır.

- **Arama Motorlarının En Üst Seviyedeki Sayfaları İndekslemesi**

Arama motorları indeksleme hacmini azaltmak için genellikle bütün Web sayfasını indekslemek yerine ana sayfadan itibaren iki veya üç seviyesini indekslemektedir. Hatta arama motorlarının genellikle sayfanın başlık bilgisi, metadata kısmı ve Body ksımında iki ya da üç paragrafı indeksledikleri belirtilmeltedir. Bu sebeple daha alt seviye bir sayfada olan bilgi bazen arama motorları tarafından gözden kaçabilmektedir.

- **Aranılan Bilginin Html İçine Gömülü Olmaması**

İstenilen bilgi HTML içine gömülü yazılardan farklı biçimde bulunabilmektedir. (Resim dosyaları, Veri tabanlarından çekilen dosyalar, PDF dosyaları vb.) Bu durum indekslenecek verinin hacmini arttırmaktadır.

- **Arama Motorlarının Kelimenin Önemi Hakkında Yorum Yapamaması**

Arama motorları kelimelerin önemi hakkında bir yorum yapamadığı için bazen ilgisiz sonuçlar arama sonucu olarak döndürülebilmektedir.

- **Meta Takılarının Bir Standardının olmaması**

Web sayfalarında <META> takısı ile belirtilen anahtar kelimelerin bir standardının olmaması arama motorlarından alınan sonuçları etkilemektedir.

Örneğin:

```
<meta name="Author" content="Sermet Soykan">
<meta name="AuthorName" content=" Sermet Soykan ">
```

Yukarıdaki örneklerin ikisinde de yazar ismi belirtmesine rağmen belirli bir standardın olmaması kullanılan özelliklerin isimlerinin farklı olmasına sebep olmakta ve arama motorlarının performansını etkilemektedir.
Arama motorları yukarıda sayılan bazı sebeplerden dolayı arama ölçütlerini daraltmak adına belirli etki alanlarında arama yapmaya imkân vermektedir. Google Scholar hizmeti bu çeşit bir aramaya örnek verilebilir. Bu hizmet ile girilen anahtar kelimeler akademik dokümanlar arasında aranmaktadır. Bu tür bir hizmet ile elde edilen sonuçların arama ölçütleri ile uygun olması sağlansa da bazı önemli dokümanların gözden kaçması gibi olumsuz yönleri de bulunmaktadır.

Web'in giderek bir bilgi çöplüğüne dönüşmesi istenilen verilere ulaşmayı gün geçtikçe zorlaştırmakta ve insanlar analiz edebileceklerinden çok daha fazla veri ile uğraşmak zorunda kalmaktadırlar. Bu nedenlerden dolayı İnternet üzerinden erişilebilen büyük miktarda ve çeşitlilikte verinin nasıl organize edilip sınıflandırılacağı endüstri firmaları, hükümet ve askeri kuruluşların karşılaştığı en büyük problemlerden biridir. Bu bilgileri bizim adımıza internetten derleyip belirli bir yapıda önümüze getirecek bir sistem üzerinde çalışmalar yapılmaktadır. Bu sistem *Anlamsal Web* (*Semantic Web*) dir. Anlamsal Web, internet'in bir sonraki aşaması olarak tanımlanmakta ve interneti artık sadece insanlar arasında bir bilgi paylaşım ortamı olmaktan çıkarmayı hedefleyip bilgisayarların da Web üzerinde bulunan bilgiyi kullanabilmesini hedeflemektedir. Bu sayede bilgi paylaşımı makineler arasında da olabilecek ve paylaşılan bu bilgi otomatik olarak işlenebilecektir.

Şu an kullanılan internet ikinci nesil internet olarak tanımlanabilir. İlk nesil internet Web'in ilk zamanlarında insanların tamamen elle "*Notepad*" gibi basit yazı editörlerinde yazdıkları HTML takılarından oluşmaktaydı. Şu an kullandığımız

internette ise bu yazım işlemi genelde "*Dreamweaver*", "*Frontpage*" gibi programlar aracılığı ile gerçekleştirilmektedir. Bu iki nesil Web için de son kullanıcı insandır. Ancak üçüncü nesil olarak adlandırılan Anlamsal Web'de ortaya konulan kaynakları bilgisayarların da kullanabilmesi amaçlanmaktadır.

Anlamsal Web'in oluşturulabilmesi için bilgisayarların bilgilerin içeriklerini de anlaması gerekmektedir. Bu sebeple sadece sayfaların sözdizimsel yapısı ile ilgili standartlar belirlemekle kalmayıp sayfaların anlamsal içeriği ile de ilgili standartlar belirlemek gerekmektedir. Bu standartlar belirlendiği zaman farklı bilgisayarlar için birlikte çalışabilirlik (*Interoperability*) kavramından söz etmek mümkün olacaktır.

2. METADATA

Farkına varmadan günlük hayat içinde metadata ile onlarca kez karşılaşılmaktadır. Bir araştırma yapmak için kütüphaneye gittiğimizi düşünelim. Binlerce kitap arasından aranılan konuyla ilgili olan kitabı bulmak oldukça zor gibi görünse de birkaç anahtar kelime ile saniyeler içerisinde aranan kitabın yerini bulmak mümkün olmaktadır. Aranılan kitabı bulmak için yazılan anahtar kelimeler aslında birer metadata diğer bir deyişle bilgi hakkında bilgiden başka bir şey değildir. Yani kitabın yeri bilgisini bulmak için yazar adı, basım tarihi, konu gibi bilgileri arama kıstası olarak girmek gerekmektedir. Kitabın yeri bilgisine ulaşma işini bu bilgileri kullanmadan halledebilmek de mümkündür. Aranan kitap kütüphanedeki kitapların hepsi taranarak ya da eskiden olduğu gibi kartlara bakılarak da bulunabilir. Ancak bu iki yöntem de metadata kullanarak yapılan aramaya göre çok daha zahmetli olup çok daha uzun vakit almaktadır. Web'i de içinde her türden bilginin olduğu çok büyük bir kütüphane gibi düşünürsek buradaki bilgiye erişmek için metadata kullanmanın kaçınılmaz olduğu ortaya çıkmaktadır.

"Meta" sözcüğünün kökeni Yunancaya dayanmaktadır ve sonraki, yanında, birlikte anlamına gelmektedir. Felsefenin bilgi ile uğraşan dalı olan Epistemolojide ise "Meta" , "hakkında" anlamında kullanılmaktadır [34].

Kısaca tanımlamak gerekirse metadata, bir bilgi kaynağını tanımlayan, açıklayan (hakkında bilgi veren) , yerini belirten ya da yönetimini kolaylaştıran bilgiye verilen isimdir. Metadata, bilgiyi yaratan kişi ile kullanan kişi arasında bir köprü gibidir. Stephanie Neil tarafından yapılmış tanım metadatayı ilginç bir biçimde ifade etmektedir: "Metadata, Web'in çöpçatanı"dır [12]. Bu tanım metadatanın Web üzerinde aranılan bilgiyi bulmayı kolaylaştırdığının en güzel ifadesidir. Web için önemi anlaşıldığı için halen Microsoft, IBM, Motorola, Netscape, Nokia gibi birçok büyük şirket metadata çatısı geliştirme konusunda çalışmalar yapmaktadır [24].

W3C'de metadata ile ilgili ilk çalışmalar 1995 yılında PICS (*Platform for Internet Content Selection*) çalışmaları ile başlamıştır. PICS, web sayfalarının içerik

bilgilerini sınıflandırmaya yarayan bir W3C tanımlamasıdır. Bu bilgiler sayesinde bir internet sayfasının içeriği (haber, argo sözcük, şiddet vb.) tanımlanarak internet siteleri için süzme işlemi uygulanabiliyordu. PICS, sabit ölçütler kullanmayıp bir değerlendirme sisteminin yaratılması için standart bir yöntem sunmaktadır. Bu sayede esnek bir yapıya sahip olan PICS ile her organizasyon kendi kıstaslarına göre içerik gruplandırması yapabiliyordu. Böylelikle firmalar kendi amaç, öncelik ve değerleri doğrultusunda süzme yapabilmektedirler.

PICS yapısında bulunan bazı eksiklikleri gören W3C bu yapıyı geliştirmek için PICS NG (*PICS Next Generation*) çalışma grubunu kurdu. Bu çalışma grubu toplandıktan sonra W3C daha önceki doküman tanımlamalarında tasarlanan alt yapının daha başka uygulamalar için de uygun olduğunu görerek bu uygulamaları daha sonra değinilecek olan W3C RDF (*Resource Description Framework*) çalışma grubu altında topladı.

2.1 Metadata Çeşitleri

Farklı amaçlar doğrultusunda farklı metadata'lar bulunmaktadır. Metadata'ları Yönetimsel (*Administrative*), Tanımsal (*Descriptrative*), Koruma (*Preservation*) ve Teknik (*Technical*) olmak üzere 4 gruba ayırmak mümkündür.

- **Yönetimsel Metadata:** Bilgi kaynaklarını yönetmek ve idare etmek için kullanılır. Yönetimsel metadata kaynağın nasıl dijital hale getirildiği, telif hakkı ve lisans bilgisi gibi verileri içermektedir.

- **Tanımsal Metadata:** Kaynağın içeriğini tanımlamak için kullanılır. Katalog kayıtları, kullanıcılar tarafından yapılan açıklamalar ya da içindekiler gibi bilgileri içermektedir.

- **Koruma Metadatası:** Bilgi kaynaklarının korunmasını düzenleme ile ilgilidir. Kaynakların fiziksel durumlarının belgelenmesi, kaynakların fiziksel ya da dijital kopyalarının korunması için yapılan işlemler gibi bilgileri içermektedir.

- **Teknik Metadata:** Bir sistemin nasıl işlediğini göstermek için kullanılmaktadır. Yazılım ve donanım dokümantasyonu, güvenlik ve kimlik denetimi (*Authentication*) , şifreler vb. bilgileri içermektedir.

2.2 Metadata Kullanım Amaçları

Metadata, verinin tanımlanması, transferi ve taranması gibi birçok önemli amaca hizmet etmektedir. Ek içerik sağlayarak dokümanların daha anlamlı, erişilebilir ve kullanışlı olmasını sağlamaktadır. Metadata genel olarak kaynakların tanımlanması ve bulunması için kullanılıyor olarak algılansa da metadata'ların dijital kaynakların yönetiminde de çok önemli bir role sahip oldukları yadsınamaz bir gerçektir.

Dijital bilgi sistemlerinde metadata aşağıda belirtilen farklı rollere sahiptir.

- **Erişimde Kolaylık:** Kaynağı tanımlama yönünden zengin bir metadata ile bulunmak istenen kaynağa ulaşmak büyük ölçüde kolaylaşmaktadır. Dijital bilgi sistemleri ve Dublin Core (DC), Text Encoding Initiative (TEI) gibi profesyonel topluluklar tarafından geliştirilen standart hale getirilmiş metadatalar ile kolleksiyon ya da tek bir kaynak için arama yapmak oldukça kolaylaşmaktadır.

- **Birlikte işlerlik:** Bir kaynağın metadatalar ile tanımlanması, o kaynağın insanlar tarafından anlaşılmasını kolaylaştırdığı gibi bilgisayarlar tarafından anlaşılmasını da kolaylaştırmaktadır. Birlikte işlerlik; farklı donanım, işletim sistemi ve veri yapısına sahip sistemlerin bilgiyi hatasız olarak değiş tokuş edebilmelerine imkân vermektedir. Metadata planları, transfer protokolleri kullanarak farklı ağlarda bulunan kaynakları kolaylıkla araştırmak mümkündür. Z39.50 protokolü farklı sistemler arasında arama yapmaya, *Open Archives Initiative* (OAI) protokolü metadata hasat edilmesine yardım etmektedir [14].

- **Çok Sürümlülük (*Multi-Versioning*) :** Nesneler bilgisayar ortamına doğrudan dijital ortamda yaratılarak (Word, Excel vb.) ya da dijital hale getirilerek (tarayıcı vb.) olmak üzere iki şekilde girerler. Aynı nesnenin çoklu sürümleri kaynağı koruma, araştırma, dağıtım ya da ürün geliştirme amacı ile yaratılmış olabilir. Kaynağı yaratan kişi bu amaç hakkında bilgi vermek için bazı yönetimsel ve tanımsal metadatalar oluşturabilir.

- **Koruma:** Sayısal veriler kolayca bozulabilmekte ve değiştirilebilmektedir. Bu sebeple kaynakların korunması gerekmektedir. Dijital ortamda yaratılan bir bilgi yaşam döngüsüne farklı bilgisayar yazılım ve donanım kuşakları arasında devam edecekse ya da yeni bir sistem üzerine yüklenecekse, bu bilgilerin farklı sistemlerde kullanılmasına imkân verecek metadatalara ihtiyaç bulunmaktadır. Teknik, Tanımsal ve Koruma metadataları dijital bir bilginin nasıl yaratıldığı ve diğer bilgi nesneleri ile ilişkileri konusunda bilgileri tutmaktadır. Bu sebeple yaratılan bir dijital bilginin erişilebilir ve anlaşılabilir olması için metadatalarının da korunması gerekmektedir.

2.3 Metadata Özellikleri

Metadataların en önemli ortak özellikleri semantik, söz dizimi ve yapıdır. Semantik, metadata elementlerinin tip ve içeriğini belirtmektedir. Söz dizimi, içeriğin belirli dilbilgisi kurallarına göre yapılandırılmasıdır. Standart ise metadataları herkesin anlayıp yorumlayabileceği ortak bir yapıya kavuşturma anlamına gelmektedir. Standartlar, ağlar arası iletişim, taşınabilirlik ve yeniden kullanım için gereklidir. Standartlar sayesinde, belirli bir ifade ile ne taşındığı bilgisine akıllarda bir soru işareti kalmadan ulaşılmış olur. Standartlar belirli bir düzeni zorunlu kılarak daha düzenli ve birörnek erişim sağlayıp farklı metadataların birbirleri ile çalışmasını sağlamayı kolaylaştırırlar [14]. Metadata standartları Dublin Core gibi basit sayılabilecek yapıda olabileceği gibi SGML gibi karmaşık bir yapıda da olabilir.

2.4 Metadata Standartları ve Element Kümeleri

Sayısal kaynakları, kullanımlarını kolaylaştırmak için içerik, özellik ve gereksinimlerini gösteren standart bir yöntemle etiketlemek gerekmektedir. Metadata standardı, belirli bir çeşit bilgi kaynağını tanımlamak için tasarlanmış metadata element kümesine verilen isimdir. Metadata standartları genellikle elementlerin isimlerini ve anlamlarını (elementin tanımı ya da anlamı) belirlemektedir. Metadata elementlerine verilen değerler içeriği oluşturmaktadır.

Birlikte işlerlik (*Interoperability*) ve Geliştirilebilirlik (*Extensibility*) bir metadata standardında olması gereken özelliklerdir. Bu özellikleri açıklamak gerekirse:

- **Birlikte İşlerlik**

Bu kavramı Sistem yönelimli ve Kullanıcı yönelimli olmak üzere iki farklı açıdan tanımlamak mümkündür. Sistem yönelimli tanımlamak gerekirse birlikte işlerlik, iki ya da daha fazla sistemin bilgi değiş tokuşu yapabilmesi ve değiş tokuş edilen bu bilgileri herhangi özel bir işlem yapmadan kullanabilmeleri anlamına gelmektedir.

Kullanıcı yönelimli olarak tanımlamak gerekirse birlikte işlerlik, iki ya da daha fazla sistemin bilgilerini direk olarak değiş tokuş edebildikleri ve değiş tokuş edilen bu bilgilerin kullanıcılar açısından tatmin edici olduğu bir durum anlamına gelmektedir [14].

Metadata açısından bakılacak olursa, birlikte işlerlik farklı metadata standartlarının birbirlerinin sahip oldukları verileri anlamaları anlamına gelmektedir. Halen çok farklı metadata standartları kullanıldığından istenen bilgiye ulaşmada farklı standartlar arasından arama yapılması kaçınılmazdır. Bu sebeple birlikte işlerlik kavramı büyük önem kazanmaktadır. Metadatalarda birlikte işlerlik iki seviyede olmaktadır. Federasyon (*Federation*) ve Hasat etme (*Harvesting*).

- **Federasyon**

Bu modelde istekler çeşitli servis sağlayıcılar tarafından karşılanır. Bu servis sağlayıcılar sorguları uzakta bulunan metadata sağlayıcılarına iletirler. Bu metadata sağlayıcılarından aldıkları sonuçları ise istemcilere gönderirler. Bu yöntem bir çeşit farklı-sistem aramasıdır. Bu arama en yaygın olarak kullanılan mekanizma daha sonra bahsedilecek olan Z39.50 protokolüdür.

- **Hasat Etme**

Bu yöntemde istemci isteği bir tek servis sağlayıcı tarafından yürütülür. Bu servis sağlayıcı metadataları farklı metadata sağlayıcılarından almaktadır. *Open Archives Initiative* (OAI) yaklaşımı bu yöntemde yaygın olarak kullanılmaktadır.

- **Geliştirilebilirlik**

Geliştirilebilirlik, çekirdek element setinin yeni eklemelerle geliştirilmesini sağlayarak daha kesin tanımlamalara imkân vermektedir.

Herhangi bir bilgiye ulaşılmasını kolaylaştırmak isterken o bilgiyi tanımlayan metadata elementlerinin çok iyi seçilmesi gerekmektedir. Uygun olmayan bir tanımlayıcının seçilmesi faydalı metadataların kullanılmasını engelleyerek anlamsal bazı boşluklar yaratacak ve bu anlamsal boşlukları doldurmak için yeni tanımlayıcıların kullanılmasını zorunlu kılacaktır. Bu nedenle kullanılacak standardı belirlemeden önce bazı kıstasların göz önünde bulundurulması gerekmektedir. Buna göre:

- Elde edilmek istenen bilgiyle metadatanın hangi parçasının ilgili olduğu belirlenmelidir. Metadatanın geliştirilmesi ve idame ettirilmesinde hali hazırdaki ihtiyaçları karşılamak ile gelecekteki ihtiyaçları da göz önünde bulundurmak arasında bir seçim yapmak gerekmektedir.

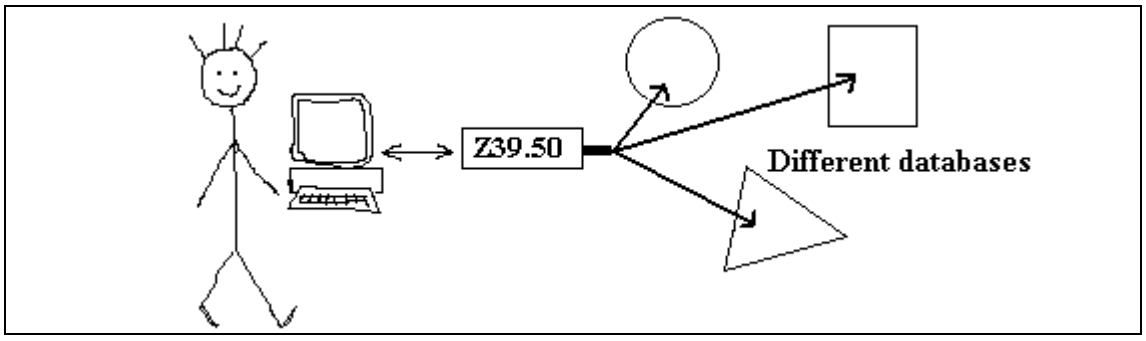
- Uygulanacak metadata şemalarının en son sürüm olması gereklidir.

2.5 Yaygın Kullanılan Protokoller

Birlikte işlerliğin sağlanabilmesi için metadataların farklı sistemler arasında gönderilmesi gerekmektedir. Bu gönderim işlemi bazı protokoller aracılığı ile yapılmaktadır. Bu protokollerden en çok kullanılanlar Z39.50 ve OAI dir.

2.5.1 Z39.50 protokolü

Z39.50, bilgiyi uzak bilgisayar veri tabanlarından arama ve getirmeye yarayan istemci-sunucu tabanlı bir protokoldür. Bu protokol bilgisayar sistemleri arasında bilginin paylaşılabilmesi için geliştirilmiştir. Z39.50 protokolü aramanın birden fazla sunucuya gönderildiği federatif sorgularda sıkça kullanılmaktadır. Sorgu sonucunda farklı kaynaklardan elde edilen sonuçlar toplanıp gereksiz bilgiler elenerek istemciye sunulmaktadır. Z39.50 protokolü ANSI ve ISO tarafından standart olarak kabul edilmiştir. Şekilde Z39.50 protokolü ile federatif bir aramanın nasıl yapıldığı gösterilmektedir.

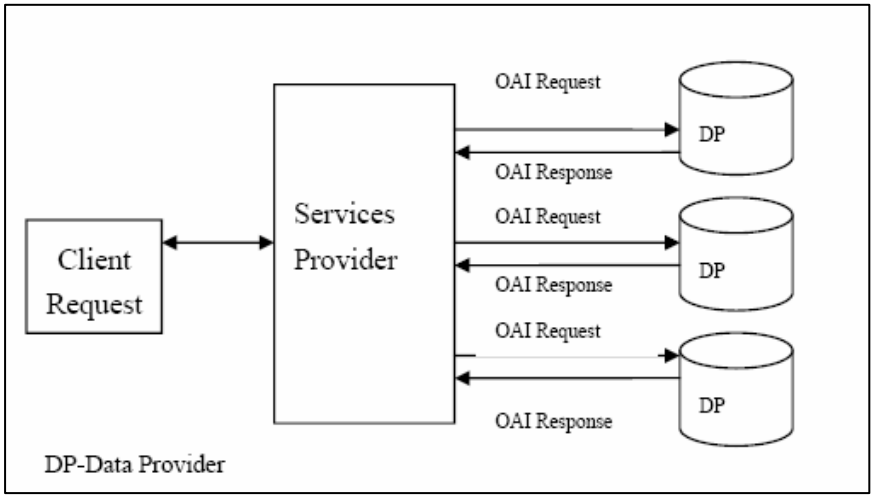


Şekil 2.1: Federatif arama için Z39.50 mekanizması

Çok kullanılan bir protokol olmasına rağmen Z39.50 veri geri getirmede tutarsızlık, zaman ve hız gibi konularda bazı problemlere sahiptir.

2.5.2 Open Archive Initiative (OAI) protokolü

OAI-PMH (*Open Archives Initiative Protocol for Metadata Harvesting*) Open Archives Initiative tarafından metadata hasat etmek (toplamak) için geliştirilmiş bir protokoldür. Protokol genellikle OAI Protokolü olarak adlandırılmaktadır. Halen 2002 yılında güncellenen 2.0 sürümü kullanılmaktadır. Piyasada baskın pozisyonda bulunan Google ve Yahoo gibi arama motorları daha fazla kaynağa erişebilmek için OAI protokolünü kullanmaktadırlar.



Şekil 2.2: OAI Servis Sağlayıcı ve OAI Veri Sağlayıcı arasındaki İlişki ve Metadata Hasat Etme İşlemi

OAI protokolünde metadatanın transferi şekil de görüldüğü gibi servis ve veri sağlayıcı arasında yapılan istek (*Request*) ve yanıtlara (*Response*) göre gerçekleştirilmektedir. Servis ve veri sağlayıcılar arasında iletişim HTTP işlemleri ile ve Get ve Post metotları ile sağlanmaktadır.

2.6 Metadatayı Yapılandırmak

Belirli bir amaç için tasarlanan metadata element kümesine “Metadata Şeması” denilmektedir. Element, bir değere sahip olan, önceden tanımlanmış karakter dizisi (*String*) ya da etikete verilen isimdir. Basit bir örnek ile elementi Author=”Sermet Soykan” şeklinde gösterebilmek mümkündür. Bu gösterimde “Author” önceden tanımlanmış karakter dizisini, “Sermet Soykan” ise bu karakter dizisine verilen değeri göstermektedir. Elementlerin tanımları ya da anlamları şemanın semantiğini oluşturmaktadır. Bu elementlere verilen değerlerle de içerik oluşturulmaktadır. Metadata şemaları genel olarak elementlerin isimlerini ve anlamlarını belirtmektedir. İsteğe bağlı olarak, içeriğin nasıl formüle edileceği (Örneğin Başlık bilgisinin nasıl belirtileceği) , içeriğin gösterim kuralları (Büyük-Küçük harf gibi) ve izin verilen içerik değerleri (Kontrollü bir sözlükten seçilen kelimeler gibi) kuralları da belirleyebilmektedir.

Bütün bunlara ek olarak elementlerin ve değerlerinin nasıl yazılacağını belirleyen söz dizim kuralları bulunmaktadır. Bazı şemalar bu söz dizim kurallarına sahip değildir. Böyle şemalara söz dizimden bağımsız şemalar denir. Metadata herhangi bir söz dizim kuralı ile yazılabilir. Halen birçok metadata şeması SGML ve XML kullanmaktadır.

2.7 Metadata Çatıları

Web sitesi tasarlayanlar metadatalarını genellikle HTML’in “META” takıları arasında belirtmektedir. Bir “META” takısı İsim (*Name*) ve İçerik (*Content*) özelliklerinden oluşmaktadır. Burada içerik özelliğinin değeri Web sitesini tasarlayan kişi tarafından belirlenmektedir. Aşağıda örnek bir “META” takısı tanımlama gösterilmektedir.

Örnek:

<META name= “description” content= “Bu makalede Metadatanın önemi vurgulanmaktadır.”>
<META name = “keywords” content = “Metadata, Metadata extraction, Dublin Core” >

Yukarıda belirtildiği gibi “META” takıları arasına eklenen bu bilgiler tamamen bu bilgiyi yaratan kişi tarafından eklenmektedir. Bu kişilerin çoğunun metadata yaratma ya da ekleme konusunda fazla bilgisi olmadığı düşünüldüğü için yaratmak için özel bilgi ve yetenek gerektirmeyen metadata formatlarına ihtiyaç doğmuştur. Bu formatlardan bazıları:

1. MARC (MAchine Readable Cataloging)
2. EAD (Encoded Archival Description)
3. GILS (Government Information Locator Service)
4. VRA (Visual Resources Association)
5. ONIX (Online Information Exchange)
6. TEI (Text Encoded Initiative)
7. FGDC (Federal Geographic Data Committee’s Content Standart for Digital Geospatial Metadata- CSDGM)
8. RDF (Resource Description Framework):
9. Dublin Core

2.7.1 MARC (Makinaca Okunabilir Kataloglama)

MARC (*MAchine-Readable Cataloging*), 1960’ların başlarında A.B.D Kongre Kütüphanesi’nde geliştirilmiş olan ve kütüphaneciler tarafından çok kullanılan bir standartlar kümesidir. MARC, bibliyografik künye bilgilerinin bilgisayarlarda anlaşılır biçimde tutulabilmesi için geliştirilmiştir. En yaygın MARC standartları USMARC, UKMARC ve UNIMARC'tır. MARC standardında kullanılan veri elementleri bugünkü kütüphane kataloglarının temellerini oluşturmuştur.

2.7.2 EAD (Şifrelenmiş Arşivsel Tanımlama)

EAD (*The Encoded Archival Description*), 1993 yılında California Üniversitesi'nde oluşturulmuştur. EAD'nin oluşumunda Amerikan Arşivciler Kurumu (*Society of American Archivists*) ve Kongre kütüphanesi'nin katkıları olmuştur.

EAD, bilgileri bulmayı kolaylaştırmak için oluşturulmuş bir XML standardıdır. Bu yapı XML standardı ile oluşturularak bilgisayarlar tarafından okunması ve işlenmesi kolaylaştırılmış böylece kütüphane, müze vb. yerlerde bilgi koleksiyonlarını araştırmak ve düzenlemek büyük ölçüde kolaylaşmıştır.
EAD standardı XML'e dayandığı için EAD dokümanlarının da birer DTD'leri bulunmaktadır. DTD ile hangi elementleri kullanmanın zorunlu olduğu ve doğru yuvalanma gibi kurallar belirtilmektedir.

EAD özel ve büyük kolleksiyonları olan akademik kütüphanelerde yaygın olarak kullanılmaktadır. Halen EAD'nin EAD 2002 sürümü kullanılmaktadır [30].

2.7.3 GILS (Hükümet Bilgi Yerleştirme Hizmeti)

GILS (*Government Information Locator Service*), ABD'de halkın hükümet bilgilerine erişebilmesini kolaylaştırmak için geliştirilmiş bir hizmettir. 1995 yılında *Paper Reduction Act* (Kâğıt Üzerindeki Çalışmaları Azaltma Hareketi) sonucu ortaya çıkmıştır. Bilgi olarak nitelendirilen her şey bu standart ile kullanılabilmektedir. GILS resmi olarak metadata öğelerini tanımlamayıp, sunum ve sözdizimi için kurallar içermektedir.

GILS'in amacı elektronik ve elektronik olmayan hükümet kaynakları için yüksek seviyede yerleşim kayıtları sağlamaktı. GILS bibliyografik tanımlamadan ziyade elde edilebilirlik ve bilginin dağıtılabilirliği üzerinde durmaktadır. Bu nedenle bir GILS kaydında veri öğeleri (ad, dağıtanın adı, işlem sırası... vb.) bulunabilir. Ancak bazı örgütler GILS'i bireysel parçalar içinde kullanırlar.

2.7.4 VRA (Görsel Kaynaklar Derneği Çekirdek Kategorileri)

VRA (*The Visual Resources Association Core Categories*); bina, fotoğraf, resim ve heykel gibi görsel materyalleri tanımlamak için geliştirilmiş metadata öğe kümesidir.

Görsel materyaller beraberlerinde orijinal çalışmanın fotoğrafını ya da slaytını da bulundurabilirler. Bu nedenle bu materyallerin metadatası kaynakların çoklu tanımlarını da birleştirmek zorundadır.

VRA'nın 4.0 sürümü 19 metadata öğesi içermektedir. Bunlar; Kayıt tipi, tip, yapıt adı, araç, ölçüleri, materyal, teknik, yaratıcı, tarih, yer, kimlik numarası, stil süresi, kültür, dönem, konu, ilişki, tanım, kaynak ve haklar [31].

Dublin Core benzeri bir yapı ile oluşturulan VRA, görsel malzemeleri tanımlamak için kullandığı elementleri XML şema yapısı ile veri standardı haline getirmiştir.

2.7.5 TEI (Metin Şifrelenme Teşebbüsü Başlığı)

TEI (*The Text Encoded Initiative Header*), sayısal ortamda bulunan metinlerin gösterilmesi için bir standart oluşturmaya çalışan enstitü ve projelerin oluşturduğu bir topuluktur.

Ana çalışma alanı bilgisayarlar tarafından okunabilen metinlerin şifrelenme yöntemleri hakkında yönergeler belirlemektir. Bu metinler beşeri bilimler, sosyal bilimler ve dilbilimi konusunda olmaktadır. TEI, roman, oyun, şiir gibi öncelikle insan bilimleri alanında elektronik metinlerin işaretlenmesi için rehberler geliştirmek amacıyla başlatılmış uluslararası bir projedir.

TEI temel olarak bir metnin anlam ifade eden bütün öğelerini şifrelemektedir. Roman, şiir gibi eserlerde en önemli öğe metin olsa da bazı durumlarda metinin anlamını güçlendirmek ve anlamı güçlendirmek için görsel bazı eklentiler yapılmaktadır. Örneğin önemli metinlerin altını çizmek ya da bazı kelimeleri yana yatık olarak yazmak gibi. TEI, metinde bulunan bu anlamsal yönleri ayırt etmeyi sağlamaktadır. TEI ilk zamanlarda SGML ile oluşturulmuş ancak son zamanlarda XML yapısı benimsenmiştir.

Bir çalışma metninin nasıl şifreleneceğini belirlemek için TEI Rehberi vardır. Ayrıca TEI bir başlık kısmını belirtir, kaynağın içine gömülmüş, çalışma hakkında metadata içerir.

2.7.6 ONIX (Çevirim-içi Bilgi Değişimi)

ONIX (*Online Information Exchange*) , elektronik olarak tanımlanmış kitaplar için tekbiçimliği sağlamak amacıyla oluşturulmuş bir standarttır. Bu standart yayıncıların kitapları hakkında bilgileri satıcılara sunmak için oluşturulmuştur. ONIX, kitap bilgilerinin farklı teknik altyapıya sahip kurumlar arasında iletilmesine imkân vermektedir [32].

ONIX, XML esaslı bir metadata tablosudur. Bu standart ile kitapların:

- Başlık
- Yazar
- ISBN numarası
- Fiyat ve temin durumu
- Tanıtıcı yazılar ve Eleştiriler
- Bölgesel Haklar

gibi özellikleri hakkında bilgi sahibi olmak mümkündür.

ONIX, çevirim-içi kitap satışı ve bu kitapların resimleri, cilt kapakları, gözden geçirme notları ve benzeri bilgiler olmaksızın kitabın satışının

gerçekleşemeyeceğinin anlaşılması sonucu bu alandaki bilgileri ve ticari bilgi gibi promosyon amaçlı bilgi ve değerlendirme içeren kayıtları oluşturmak için gerekli öğelere sahiptir.

ONIX, kitap satıcıları ve dağıtıcıları kitap satış bilgi iletişimini sağlamaya odaklanmasına rağmen diğer yayın türlerini, medya, dergileri, dergi makalelerini, konferans notları ve elektronik kitapları içerecek şekilde geliştirilmektedir.
E-kitap endüstrisi metadata'nın daha etkin kullanılmasını elektronik kitap dağıtımında daha önemli olacağını konuşmaktadır [11].

2.7.7 FGDC (Federal Coğrafi Veri Komitesi'nin Sayısal Coğrafik-Uzaysal Metadata İçin İçerik Standardı)

Federal Coğrafi Veri Komitesi (*FGDC*) tarafından coğrafik bilgi sistemlerinde kullanılmak üzere metadata içerik bilgisini belirlemek ve ilgili konulardaki kavramlar için ortak bir terminoloji yaratmak amacıyla geliştirilmiştir

Coğrafi metadata dünya üzerinde herhangi bir yerde açık veya kapalı bir şekilde coğrafi izdüşümü olan nesneleri belirtmek için kullanılmaktadır. Coğrafi bir metadata kaydı Başlık, Öz ve Yayın bilgisi gibi veriler içerebileceği gibi Coğrafi uzantı ve İzdüşüm Bilgisi gibi coğrafi elementler de içerebilir [33]. Bu yapı da bu konuda bir standart oluşturmak amacıyla geliştirilmiştir.

Anlatılan bu standartların dışında metadatanın işlenmesiyle ilgili bir model olan RDF, metadata çatı yapılarında anlatılacaktır.

2.8 Metadata Çatı Yapıları

Metadata'lar arası birlikte işlerlik, farklı setlerin etkileşimli çalışabilecekleri çatı yapılarla sağlanabilmektedir. Bu bağlamda metadatanın alt yapısı ile ilgili çalışmalar olarak adlandırabilecek Kaynak Tanımlama Çatısı (Resource

Description Framework-RDF) ve Warwick Çatısı benzeri çalışmalar, metadatalar arası birlikte işlerliği ve etkileşimi sağlamayı hedefleyen yapılardır.

2.8.1 Kaynak tanımlama çatısı (Resource Description Framework)

Önceki bölümlerde bahsedilen PICS ve PICS-NG'nin çalışmalarını takiben W3C, Web için daha genel metadata tanımlama modeli olarak RDF çalışmasını başlatmıştır. RDF'in ortaya çıkmasının nedeni daha önceki çalışmaların, Internet kaynaklarının tanımlayıcı bilgilerinin oluşturulmasında gereksinimleri karşılayacak işlevsel yeterliliğe sahip olmamasıdır. RDF, yapılandırılmış metadata'nın yeniden kullanımı, değişimi ve kodlanmasını sağlayan bir alt yapı sistemi şeklinde karşımıza çıkmaktadır [1].

Resource Description Framework, 1999'un başlarında W3C tarafından geliştirilmiş evrensel bir metadata taşıyıcıdır. En genel anlamıyla bir Web kaynağının içerik bilgilerini tanımlamaya yarayan XML tabanlı bir modeldir. Burada bahsedilen kaynak bir web sayfası olabileceği gibi bir web sitesinin tamamı ya da Web üzerinde bulunan bilgi içeren herhangi bir öğe de olabilir [24]. Bu sebeple anlaşılmada kolaylık olması bakımından Web ile erişilebilen bir dosya olarak düşünmek mümkündür. Bir Web sitesinin metadatasını ifade etmekle genel olarak bir veriyi ifade etmek arasında bir fark yoktur. Bu sebeple RDF, Web içeriğini tanımlamak dışında her türlü kaynağı tanımlamak için kullanılabilir [28]. RDF, bir dokümanın içeriğini göstermek yerine, yazar, tarih ve tür gibi harici bilgilerini tutmak için kullanılır. RDF bizlere bilgiyi makinelerin de anlayabileceği bir yapıda saklamak için fırsat sunan bir yapıdır. Bu sayede farklı uygulamaların metadataları değiş tokuş edip kullanabilmesine imkân sağlanır. Metadataların uygulamalar arasında değiş tokuş edilmesini, aranmasını ve kataloglanmasını kolaylaştırıp metadataların birlikte işlerliğine imkân verilmiş olur.

2.8.2 Warwick çatısı

1996 yılında Warwick Üniversitesinde, UKOLN (*UK Office for Library and Information Networking*) ve OCLC'nin (*Online Computer Library Center*) ortaklaşa gerçekleştirdiği ikinci Dublin Core Metadata Çalıştayı sonucunda ortaya çıkan, metadatalar arası değişim ve birlikte çalışabilirliğe yönelik farklı üst veri paketlerini birleştirici özelliği olan bir uygulamadır. Söz konusu yapı iki temel bileşenden oluşmaktadır. Bunlar; farklı formatlarda üst veri bilgilerinin yer aldığı "paketler" ve paketleri bir araya toplayan "taşıyıcı" yapılardır [1].

2.9 Standartlar Arası Eşleştirme Çalışmaları

Metadatanın uluslararası işlemselliği ve değişimi ancak eşleştirme çalışmaları ile sağlanabilir. Bilginin farklı sistemler ve kaynaklardan alınarak derlenebilmesi anlamsal bir uyumluluk ile mümkün olmaktadır. Farklı terminolojilerin kullanımı ve standart şemalarda yapılan değişiklikler disiplinler ve sektörler arası bilgi bulmayı zorlaştırmaktadır. Kullanıcının aradığı veri ile ilgili bütün bilgileri elde edebilmesi için farklı metadata şemaları arasında uyum olması gerekmektedir. Uyumu sağlama konusunda birçok farklı öneri sunulmuş olsa da en büyük dikkati "Eşleştirme" (*Mapping*) yaklaşımı çekmiştir. Doerr eşleştirmeyi, "Birbirlerine yaklaşık olarak eşit olan terim, içerik ve hiyerarşik ilişkilerin belirlenmesi" olarak tanımlamaktadır [8]. Halen birçok farklı metadata standardı bulunmaktadır. Bu standartlar, farklı ihtiyaçlar doğrultusunda farklı kullanıcılara hizmet etmektedir. Değişik standartlara göre oluşturulmuş bu metadatalar arasında ilişki eşleştirme çalışmaları ile kurulmaya çalışılmaktadır. Bu eşleme Yayageçidi (*Crosswalk*) denilen yapılarla sağlanmaktadır. Yayageçitleri, farklı şemaları kullanan veri tabanları arasında arama yapmaya izin veren yapılardır.

Eşleştirme çalışmalarındaki gelişmeler metadatalar arasında birlikte işlerliği arttıracağından kullanıcıların aradıkları veri ile ilgili daha fazla sonuç elde etmeleri sağlanmış olacaktır.

3. DUBLIN CORE

Dublin Core standardı, “İnternet ya da Web üzerinde aranan bilgiyi bulmak neden bu kadar zor?” sorusuna bir cevap olarak ortaya çıkmıştır [14]. “Dublin Core Meta Data Element kümesi”, kaynakları tanımlamak için 15 elementten oluşan ve farklı etki alanındaki kaynakları tanımlamak için oluşturulmuş bir standarttır. Bu standart ile ilgili çalışmalar 1995 yılında ABD’nin Ohio eyaletinin Dublin kentinde yapılan bir çalışma ile başlamıştır. Dublin ismi çalışmanın başladığı şehirden gelmektedir. Core (Çekirdek) ismi ise element kümesinin basit fakat genişletilebilir olduğunu belirtmektedir. Dublin Core; video, ses, resim, yazı ve web sayfaları gibi dijital dokümanları tanımlamada kullanılmaktadır. Dublin Core ile ilgili çalışmaları *Dublin Core Metadata Initiative* isimli kuruluş gerçekleştirmektedir. Bu kuruluşun çalışma amaçları:

- Farklı alanlardaki kaynakları arama ve bu kaynaklara erişmek için metadata standartları geliştirme
- Metadata kümelerinin birlikte işlerliğini sağlamak için çatı (*Framework*) belirleme
- Bu çatılarla çalışacak topluluklara veya konulara özel metadata kümelerinin geliştirilmesini kolaylaştırmaktır.

Basit Dublin Core Meta Data Element Set (DCMES) 15 metadata elementinden oluşmaktadır. Bu elementler:

1. Title (Yapıt Adı)
2. Creator (Yaratan / Yazar)
3. Subject (Konu / Anahtar Kelimeler)
4. Description (Tanım)
5. Publisher (Yayınlayan)
6. Contributor (Katkıda Bulunanlar)
7. Date (Tarih)
8. Type (Tip)

9. Format (Biçim)
10. Identifier (Tanımlayan / Tanımlayıcı)
11. Source (Kaynak)
12. Language (Dil)
13. Relation (İlişki)
14. Coverage (Kapsam)
15. Rights (Haklar ve Yönetimi)

Şekil 3.1'de bu element kümesinin gruplandırılmış hali gösterilmektedir.

Şekil 3.1: DCMES Elementlerinin gruplandırılması

Şekil 4'te gösterildiği gibi bu 15 öğeyi 3 grup altında sınıflandırmak mümkündür. Bu sınıflandırmaya göre bazı öğeler kaynağın içeriğiyle, bazı öğeler kaynağın entellektüel özellikleriyle, bazıları ise kaynağın seçici özellikleriyle ilgilidir:

1. İçerik : 7 öğe (kapsam, tanım, tip, ilişki, kaynak, konu ve yapıt adı)
2. Entelektüel: 4 öğe (katkıda bulunanlar, yaratıcı, yayınlayan ve haklar)
3. Yapısal (Şekilsel): 4 öğe (tarih, format, tanımlayıcı ve dil)

Dublin Core yapısında bulunan elementlerin hepsi seçime bağlıdır, tekrarlanabilir ve sıralamaları önemsizdir. İhtiyaç duyulduğu takdirde elementlerin içerikleri

nitelendiriciler (*Qualifier*) ile daha iyi açıklanılabilir. Nitelendiriciler, bir elementin içeriği hakkında ek bilgiler veren yapılardır. Örneğin "*Date*" elementi "Date (scheme=ISO 8601)=2007–03–12" şeklinde kullanılırsa, ISO 8601 tarih yapısını YYYY-MM-DD şeklinde gösterdiği için tarihin 12 Aralık 2007 değil 12 Mart 2007 olduğu rahatlıkla anlaşılabilir.

Dublin Core elementleri birçok kaynağı tanımlamak için yeterli olsa da bazı durumlarda ek tanımlayıcılara ihtiyaç duyulabilir. Böyle durumlarda bu temel elementlere başka metadata tanımlayıcıları ekleyerek daha karmaşık tanımlayıcılar oluşturmak da mümkündür. Dublin Core elementleri HTML, XML ve RDF içerisinde taşınabilmektedirler.

Dublin Core standardının oluşturulduğu ilk çalışma toplantısında, Dublin Core ile ilgili bazı ilkeler belirlenmiştir. Bu ilkelerin belirlenmesindeki amaç Dublin Core standardını mümkün olduğunca basit, anlaşılabilir ve esnek tutmaktır. Bu ilkeler: İçsellik, Genişletilebilirlik, Söz dizimden Bağımsızlık, Seçime bağlılık, Tekrarlanabilirlik ve Değiştirilebilirlik dir.

- **İçsellik (*Intrinsicality*)**

Dublin Core standardı kaynağın içinde bulunan özellikleri tanımlamaktadır. Yani Dublin Core ile anlatılan bilgilere sadece kaynak üzerinde bir çalışma yaparak ulaşmak mümkündür. Dış kökenli (*Extrinsic*) veri, kaynağa erişim gibi kaynağın içeriği ile direk erişilemeyen bilgileri belirtmektedir. Bu bilgiler gerekli olduğu takdirde element kümesi genişletilerek tanımlanabilir.

- **Genişletilebilirlik (*Extensibility*)**

Genişletilebilirlik sayesinde dış kökenli bilgiler tanımlandığı gibi element kümesinde ait olmayan verileri de tanımlamak mümkündür. Dublin Core standardının sürekli geliştirildiği düşünüldüğünde yeni elementlerin eklenerek

yapının genişletilmesi eski elementleri terk etmeyerek geriye doğru uyumluluğu (*Backward Compatibility*) da sağlamaktadır.

- **Söz dizimden Bağımsızlık (*Syntax Independence*)**

Dublin Core standardı farklı uygulama ve platformlarda kullanılmak üzere geliştirilmiştir. Sözdizime bağlı kalınması element setinin farklı platformlarda kullanılmasına engel teşkil edebilmektedir.

- **Seçime Bağlılık (*Optionality*)**

Dublin Core yapısında elementlerin seçime bağlı olmasının iki sebebi bulunmaktadır. Bunlardan birincisi bütün elementler her kaynağa uygun olmayabilmektedir. İkinci sebep ise metadata yapısı hakkında fazla bilgi sahibi olmayan birisinin kendini bütün elementleri kullanmaya mecbur hissetmeyerek sadece mantığına uyan elementleri kullanması hedeflenmektedir.

- **Tekrarlanabilirlik (*Repeatability*)**

Dublin Core yapısındaki bütün elementleri herhangi bir kısıtlama olmaksızın istenildiği kadar tekrar etmek mümkündür. Örneğin 5 yazarı olan bir kaynakta "*Author*" elementi beş kere yazılabilir.

- **Değiştirilebilirlik (*Modifiability*)**

Dublin Core oluşturulurken bütün elementler ilave açıklamaya mahal vermeyecek şekilde oluşturulmuştur. Ancak Dublin Core standardının en önemli özelliği farklı etki alanında bulunan insanların kendi içeriklerini belirlenen 15 elementle tanımlayabilmeleridir. Bu sebeple bazı durumlarda anlam karmaşasını önlemek adına daha önce belirtilen nitelendiricileri kullanarak anlamda daraltma yapmak

mümkündür. Böylece farklı standartlar arasında eşleme (*Mapping*) yaparken ortaya çıkabilecek karmaşalar da önlenmiş olmaktadır.

Dublin Core öğeleri herkes tarafından anlaşılabilecek ve her konu alanındaki kaynakların tanımlanabilmesi için çok esnek bir yapıdadır. Bu sebeple Dublin Core öğeleri uluslararası düzeyde kabul görmekte ve desteklenmektedir. Dublin Core, kullanıcıların aradıkları kayıtları bulmaya odaklanmış ve düzenli olarak geliştirilmektedir. Tablo 3.1'de Dublin Core elementleri ve kısa açıklamaları gösterilmektedir.

Öğe	**Etiket**	**Açıklama**
Başlık	Title	Belgeyi yaratan kişinin belgeye verdiği isim
Yazar ya da yaratıcı	Creator	Belgeyi meydana getiren kişi ya da tüzel kuruluşlar
Konu ve anahtar kelimeler	Subject	Kaynağın ne ile ilgili olduğu ve belgenin konusu hakkında fikir veren belli başlı anahtar kelimeler
Tanım	Description	Kaynağın içeriğini tanımlamada yardımcı olması amacıyla kullanılır
Yayıncı	Publisher	Kaynağı yayınlayan kurum
Diğer katkıda bulunanlar	Contributor	Yazar ya da yaratıcı olarak adı geçmeyen, ama editör, çeviren vb gibi kaynağın içeriğine katkıda bulunan kişiler
Tarih	Date	Kaynağın yaratılış tarihi
Kaynak tipi	Type	Kaynağın türü (roman, şiir, teknik rapor, sözlük vb)
Biçim	Format	Kaynağın formatının ne olduğu (html, pdf vb)
Kaynak tanımlayıcı	Identifier	URL veya URN gibi kaynak adresi ya da numarası
Kaynak	Source	Elektronik kopyanın dayandığı bilgi kaynağı
Dil	Language	Kaynağın hangi dilde olduğu
İlişki	Relation	Bilgi kaynağının diğer kaynaklarla ilişkisi
Kapsam	Coverage	Kaynağın uzaysal ya da zamansal özellikleri
Hakların yönetimi	Rights	Telif hakkıyla ilgili bir ifade

Tablo 3.1: Dublin Core Elementleri

3.1 Dublin Core Elementleri

3.1.1 Katkıda bulunanlar (*Contributor*)

Bu element tanımlanan kaynak oluşturulurken katkıda bulunan bir öğeyi belirtmek için kullanılır. Bu elementin daha sonra tanımlanacak olan *Creator* elementinden farkı, *Creator* kaynağı yazıp onu baştan yaratan kişi iken *Contributor,* kaynağa düzenleme yapmak veya eklentide bulunmak gibi ikinci derecede katkılarda bulunmaktadır. Yazımında dikkat edilmesi gereken, kişilerin önce soyadı sonra adı yazılmalı, farklı kişiler virgül ile ayrılmalıdır.

Örnek:

```
<meta name="DC.contributor" content="Soykan, Sermet">
```

3.1.2 Kapsam (*Coverage*)

Bu element kaynağın uzaysal ya da zamansal özelliklerini belirtmek için kullanılır. Element kaynağın uzaysal olarak tanımlanması için kullanılırsa fiziksel bölge, koordinatlar ya da yer isimlerini kullanabilmek mümkündür. Zamansal olarak, kaynağın ne hakkında olduğu konusunda bilgi verilir. Bu element içinde kaynağın ne zaman oluşturulduğu veya kullanıma ne zaman açıldığı hakkında bilgi verilmez bu tanımlamalar için Date elementi kullanılır.

Örnek:

```
<meta name="DC.coverage" content="Kurtuluş savaşı ; 1919-
1923">
<meta name="DC.coverage" content="Türkiye ; Lat: 36 42 N
Long: 26 45 W">
```

3.1.3 Yaratan / Yazar (*Creator*)

Kaynağın oluşturulmasından birinci derece sorumlu yani kaynağı oluşturan kişiyi belirtmek için kullanılır. Kaynak yazılı bir metinse bu element bir yazarı belirtirken kaynak görsel bir öğe ise fotoğrafçı, grafikçi ya da ilüstratör olabilir. Burada dikkat edilmesi gereken kişi isimlerinin kaynakta görüldüğü sırada ve önce soyadı sonra ismi gelecek şeklinde yazılması, kişilerin virgül ile ayrılmasıdır.
Örnek:

```
<meta name="DC.creator" content="Van Gogh, Vincent">
<meta name="DC.creator" content="Soykan, Sermet">
```

3.1.4 Tarih (*Date*)

Kaynağın yaşam süreci içerisinde önemli bir tarihi ya da zaman aralığını belirtmek için kullanılır. Bu kaynağın yaratılma tarihi olabileceği gibi kullanıma açılma (siteye eklenme) tarihi de olabilir. Tarih yazılırken YYYY-MM-DD şeklinde yazılması tavsiye edilmektedir.

Örnek:

```
<meta name="DC.date" content="1972">
```

3.1.5 Tanım (*Description*)

Kaynağın tanımını içermektedir. Bu tanımda bir özet, indeks ya da düz yazı şeklinde kaynağı anlatan metin olabilir.

Örnek:

Kaynağın tanımının özet olarak yapılmış hali:

```
<meta name="DC.description.abstract" lang="tr"
content="Yazar, Dublin Core yapısının tanımı, kullanımı ve
özellikleri hakkında detaylı bilgi vermektedir. ">
```

Kaynağın tanımının indeks olarak yapılmış hali:

```
<meta name="DC.description.tableOfContents"
 content="Giriş; Kısaltmalar; Metadata nedir?; RDF nedir?
">
```

3.1.6 Biçim (*Format*)

Kaynağın dosya formatı, fiziksel ortamı ya da süresi gibi bilgiler tanımlanır. Kaynağı gösterecek yazılım ya da donanımı belirtmek için kullanılır.

Örnek:

```
<meta name="DC.format" content="text/html; 12 Kilobyte">
```

3.1.7 Tanımlayan / Tanımlayıcı (*Identifier*)

Kaynağı kesin olarak tanımlayıp benzerlerinden ayrılmasını sağlayan bir düz yazı (*String*) ya da sayıdır. Bu benzersiz yazı bir URL olabilirken sayı da kitapları tanımlamak için kullanılan ISBN numarası olabilir.

Örnek:

```
<meta name="DC.identifier" scheme="URI"
```

```
content="urn:isbn:1-56592-149-6">
```

3.1.8 Dil (*Language*)

Kaynağın yazım dilini belirtmek için kullanılır.

Örnek:

```
<meta name="DC.language" scheme="ISO639-2" content="eng">
```

3.1.9 Yayınlayan (*Publisher*)

Kaynağa ulaşılmasını mümkün kılan öğe tanımlanır. Bu bir basım evi olabileceği gibi internet üzerinden erişilmesini sağlayan servis sağlayıcı ya da bir üniversitenin bölümü olabilir.

Örnek:

```
<meta name="DC.publisher" content="Wrox Press">
```

3.1.10 İlişki (*Relation*)

Kaynakla ilgili başka bir kaynağı tanımlamak için kullanılır. Bu elementle, belirtilen kaynağın bir başka kaynağın bir parçası mı olduğu, başka bir yayının çevirisi mi olduğu ya da bir kitabın bir bölümü mü olduğu gibi bilgiler belirtilmektedir.

Örnek:

```
<meta name="DC.relation.references" scheme="URI"
 content="urn:isbn:1-56592-149-6">
```

3.1.11 Haklar ve yönetimi (*Rights*)

Kaynak üzerindeki telif hakkı bilgilerini tanımlamak için kullanılır.

Örnek:

```
<meta name="DC.rights" lang="en" content="Copyright Acme 2007 – Her hakkı saklıdır.">
```

3.1.12 Kaynak (*Source*)

Dublin Core elementleri genelde tanımlanmak istenilen kaynak hakkında bilgi içerir. Ancak *Source* elementi bir istisnadır ve kaynağın oluşturulmasına yardımcı olan kaynak veya kaynaklar hakkında tarih, oluşturan kişi vb. metadata bilgilerini içermektedir. Kaynağın hangi kaynaklardan yararlanılarak oluşturulduğunu belirtmek için kullanılır. Tanımlanan kaynak tümüyle ya da kısmen belirtilen kaynaktan yararlanılarak oluşturulmuş olabilir.

Örnek:

```
<meta name="DC.source" content="Shakespeare; Romeo and Juliet">
```

3.1.13 Konu / Anahtar Kelimeler (*Subject*)

Bu element ile kaynağın konusu belirtilir. Genelde içeriği tanımlayabilmek için anahtar kelimeler kullanılmaktadır.

Örnek:

```
<meta name="DC.subject" scheme="MESH" content="Bilgi
yönetimi; Anlamsal Web">
```

3.1.14 Yapıt adı (*Title*)

Kaynağa yazar ya da yayıncı tarafından verilen başlık bilgisini belirtmek için kullanılır.

Örnek:

```
<meta name="DC.title" content="Suç ve Ceza">
```

3.1.15 Tip (*Type*)

Kaynağın ana sayfa, roman, şiir, teknik rapor gibi kategorisini belirtmek için kullanılır.

Örnek:

```
<meta name="DC.type" content="şiir">
```

15 temel element ile ilgili çalışmalar tamamlandıktan sonra DCMES' i geliştirmeye yönelik bir çalışma başladı. Bu geliştirme çabaları tanımlanan temel elementleri daha özel hale getirmek yani anlamlarını daraltmak için yapılmıştır. Bu yeni yapıya Nitelikli Dublin Core (*Qualified Dublin Core*) adı verilmiştir.

Nitelikli Dublin Core, halen 12 elementten oluşmaktadır. Bunlar:

1. Collection
2. Dataset

3. Event
4. Image
5. InteractiveResource
6. MovingImage
7. PhysicalObject
8. Service
9. Software
10. Sound
11. StillImage
12. Text

Dublin Core'u diğer çatılardan ayıran en önemli özellikler basitlik, anlamsal birlikte işlerlik, uluslararası fikir birliği ve esnekliktir. Dublin Core oluşturulurken basitlik ve esneklik ön planda tutulmuştur. Bilgi kaynağını oluşturan kişi Dublin Core Metadata Element Set'inde bulunan elementleri kullanarak kendi kaynağı hakkında kolaylıkla metadata oluşturabilir.

Birlikte işlerlik özelliği sayesinde Dublin Core ile oluşturulan kaynak, tanımları farklı alan ve disiplinlerde de kullanılabilmektedir. Dublin Core'un metadata standardı olarak kabul görme sebeplerini toparlamak gerekirse:

- Belirlenmiş 15 öğe anlamsal olarak tarif edilmiş ve standartlaştırılmıştır.
- Her tek grup için zorunlu olmaksızın öğeler tekrarlanabilir.
- Veri kolayca oluşturulabilir.
- Oluşturulan veri diğer uygulamalarla kullanılabilir.
- Öğeler tanımlayıcı kullanılarak belirtilebilir.
- Geliştirilmesi ve uluslararası olarak yürütülmesi, OCLC (Çevrimiçi Bilgisayar Kütüphane Merkezi)'nin taahhüdü ve yakın ilişkili olduğu W3C ve diğer standardizasyon organizasyon çalışmaları tarafından garanti edilmektedir.

Dublin Core bütün bu özelliklerinden ötürü diğer metadata standartlarına göre Web üzerinde kaynak tanımlaması için potansiyel bir uluslararası standart ve metadata için web üzerinde ortak bir dil olarak görülmektedir.

Dublin Core'un olumlu özelliklerinden biri olarak görülen basitlik aynı zamanda tutarsızlık ve güven sorununu da beraberinde getirmektedir.

3.2 Dublin Core Kullanma Sebepleri

Dublin Core internet toplulukları tarafından geniş bir kabul görmüş ve de fakto internet metadata standardı haline gelmiştir. Dublin Core yapısının sunduğu avantajlar:

- Kullanılabilirlik ve esneklik
- Sunduğu elementleri internet üzerinde yayın yapmak isteyen sıradan bir kullanıcının bile kolaylıkla kullanabilmesi
- Diğer kaynak tanımlamalarının yerine almak için değil aksine onları tamamlamak için yaratılmış olması
- Bütün elementlerinin seçime bağlı olmasına rağmen hangi elementin seçime bağlı hangisinin zorunlu olduğunu belirtebilmesi
- Bütün elementlerin tekrarlanabilir olması
- İsteyen topluluklar tarafından kendi ihtiyaçları doğrultusunda geliştirilebiliyor olması

Aşağıda örnek bir DC tanımlaması gösterilmektedir.

Örnek:

```
<HTML>
<HEAD>
<TITLE>Automatic Metadata Extraction </TITLE>
```

```
<META NAME="DC.Title" CONTENT=" Otomatik Metadata
Çıkarma">
<META NAME="DC.Creator" CONTENT="Sermet Soykan">
<META NAME="DC.Type" CONTENT="text">
<META NAME="DC.Date" CONTENT="2007">
<META NAME="DC.Format" CONTENT="text/html">
</HEAD>
```

3.3 Otomatik Metadata Yaratma ve Çıkarma

Otomatik metadata çıkarma son yıllarda önemli bir araştırma konusu olmuştur. Son 20 yılda yazı analiz tekniklerindeki gelişmeler sınırlı etki alanlarında başarılı sonuçlar vermiştir. XML gibi yapılandırılmış işaretleme dilleri dokümanları insanlar için olduğu kadar bilgisayarlar için de erişilebilir hale getirmiştir. Belirli bazı bilgiler kodlanarak doküman yapısı içinde bilgisayarlar tarafından erişilebilir hale getirilmiştir. Başlık, Yazar, Yayıncı, Yayın tarihi, anahtar kelimeler ve özet gibi temel metadatalar herkesin görmesi bakımından genelde ilk sayfada bulunmakta ve genelde birörnek yapıda bulunmaktadır. Otomatik metadata çıkarma işlemi dokümanın yapı ve stiline bağlıdır.

Metadata çıkarma en genel anlamı ile metadata yaratma veya üretme işlemine verilen isimdir. İyi kalitede oluşturulan bir metadata, Web üzerine koyulan kaynağa erişimi büyük ölçüde kolaylaştırmaktadır. Metadata üretme ve yönetimi otomatik ve otomatik olmayan (insan müdahalesi) işlemlerin bir karışımı haline gelmiştir.

Web üzerindeki kaynak sayısının gün geçtikçe artması gün geçtikçe metadata'ya ihtiyacı olan nesnelerin sayısının artması anlamına gelmektedir. Bu kaynaklar için metadata üretebilecek yetkinliğe sahip insanların sayıca yetersiz ve tüm bu kaynaklar için metadata oluşturmanın maddi yükünün fazla olması otomatik metadata oluşturma işlemini zorunlu kılmaktadır.

Önceleri metadata üretme fikri yerine kaynağı hazırlayan kişilerin kendi metadatalarını oluşturması düşünülse de bu düşünce fazla kabul görmemiştir. Bu fikrin kabul görmemesindeki en önemli sebep üretilen metadataların doğruluğu hakkında olan şüphelerdir. Metadata hakkında bilgisi olmayan bir yazarın oluşturduğu metadata eksik ve kalitesiz olacaktır. Oluşturulan bu metadatalar kaynak bulma amacına tam olarak hizmet edemeyecektir.

Bu nedenle Otomatik metadata oluşturma işlemi insan odaklı yaklaşıma göre daha etkili, masrafsız, güvenilir bulunmakta ve bu işlem sonucunda kabul edilebilir sonuçlar alınmaktadır. Buna rağmen sadece bilgisayar odaklı bir yaklaşım da tek başına yeterli değildir. İnsanların bilgisayarlara göre üstün yönü olan yorum yeteneği ile bilgisayarların işlem hızlarının birleştirilmesi ile oluşturulan bütünleşik bir yaklaşım metadata üretimi için en etkin yol olarak gözükmektedir. Metadata üretiminde bu yaklaşımı benimseyen çeşitli araçlar bulunmaktadır.

Halen piyasada, farklı çıkarma yöntemlerini kullanan açık kaynak kodlu ve ticari çok çeşitli metadata üretme araçları bulunmaktadır.

3.3.1 Metadata çıkarma yöntemleri

Metadata Özütleme (*Extraction*) ve Toplama (*Harvesting*) iki önemli metadata çıkarma yöntemidir. Özütleme, Web tarayıcı tarafından görüntülenen bir kaynağın içeriğinden metadatayı bir algoritma yardımıyla alma yöntemidir. Tezimizde, gösterilen kaynağın sadece yazı kısmıyla ilgilenilecektir. Web kaynaklarında çıkarım işlemi içerik bilgisinin yoğun olarak bulunduğu HTML ya da XHTML dokümanının "Body" kısmından gerçekleştirilmektedir. Bu çıkarım işlemleri gelişmiş otomatik indeksleme ve sınıflandırma algoritmaları içerebilir. Özütleme (*Extraction*) işlemine birçok arama motorunda girilen bir ölçüte göre getirilen sayfalardan alınan cümleler örnek verilebilir. Bu cümleler sayfanın kaynağından bulunma sırasına göre doğrudan getirilmektedir.

Diğer metadata oluşturma yöntemi olan Toplama (*Harvesting*) yöntemi, HTML kaynak kodunun "*Header*" takısı arasında bulunan "Meta" takılarından oluşturulmaktadır. Bu yöntem ile toplanan metadatalar, insanlar ya da otomatik ya da yarı otomatik işlemler sonucunda programlar tarafından oluşturulmaktadır. Örneğin Microsoft FrontPage ya da Macromedia Dreamweaver programları ile yaratılan bir Web sayfasında, sayfanın oluşturulma ya da güncellenme tarihi, yazarın adı gibi bilgiler insan müdahalesi olmadan otomatik olarak sayfaya eklenmektedir. Yarı otomatik işlemde ise programlar kullanıcıya doldurması gereken bir şablon sunup kullanıcının bu şablona girdiği verileri "Meta" takılarına ya da dokümanın formatına göre uygun olan takıya çevirmekte ve oluşturduğu bu takıları sayfanın "*Header*" kısmına eklemektedir.

Yukarıda anlatılan bu iki yöntem otomatik metadata oluşturma işlemi için çok büyük bir öneme sahiptir. Bu yöntemlerden birini ya da her ikisini de kullanarak metadata oluşturan araçlar mevcuttur.

3.3.2 Metadata üretme araçları

Metadata üretme araçlarını Şablonlar, Editörler ve Üreticiler olmak üzere üç gruba ayırmak mümkündür.

3.3.2.1 Şablonlar

Şablonlar, oluşturulması ve bakımı kolay olduklarından metadata üretme alanında baskın durumdadırlar. Bu araçlar metadatanın üretim aşamalarında kullanıcıya rehberlik ederek kullanıcıyı gereksiz detaylarla uğraştırmazlar.

3.3.2.2 Editörler

Editörler de insan müdahalesi gerektirmeleri bakımından şablonlara benzemektedir. Editörler metadata yaratma işleminde sözdizimsel anlamda

kullanıcıya yardımcı olurlar. Genel anlamda editörler Web Form yapısında bulunmaktadır. Bu yapı ile kullanıcının metadata oluşturmasını kolaylaştırmaktadırlar. http://dublincore.org/tools/ adresinde editör ve şablonların kısmi bir listesi bulunmaktadır.

3.3.2.3 Üreticiler

Metadata oluşturmak için en çok kullanılan araçlardandır. Üreticiler ile metadata üretmek için öncelikle metadatası çıkarılacak olan kaynağın URL (*Uniform Resource Locator*) ya da PURL'si (*Persistent Uniform Resource İdentifier*) ya da Web adresi programa girilir. Daha sonra nesnenin içeriği dolayısı ile kaynak kodu belirli bir algoritmaya göre taranarak istenilen metadata kısımları çıkarılır. DC-dot, Describthis ve Klarity metadata üretici programlara örnek olarak verilebilir.

- **DC-dot**

DC-dot açık kaynak kodlu ve GNU Public Lisansı altında değiştirilebilen bir metadata üreticidir. UKOLN (UK Office for Library and Information Networking) tarafından Bath Üniversitesinde geliştirilmiştir. Program Dublin Core metadatası oluşturmanın yanında USMARC, SOIF, IAFA/ROADS, RDF gibi farklı metadata şemaları oluşturabilmektedir. Metadata üretilmesi istenen URL programın metin kutusu kısmına yazılıp "Gönder" düğmesine basılarak program çalıştırılmaktadır. Program girilen URL'nin kaynak kodundaki "Meta" takıları arasından "Title", "Keywords", "Description" ve "Type" kısımlarını almaktadır. Eğer girilen URL'de "Meta" takıları yoksa program çapalar, koyu yazılmış kısımlar ve bağlantıları inceleyerek otomatik olarak "Keywords" takısı yaratmaktadır.

- **Describethis (DCS)**

DescribeThis (http://www.describethis.com), çevrimiçi kaynaklardan metadata çıkarmak için tasarlanmış bir internet tabanlı hizmettir. Programın kullanıcı dostu bir ara yüzü bulunmaktadır. Metadata oluşturulması istenen kaynağın URL'si ana

ekranda girilmektedir. Sonuç, XML, XHTML ya da RDF olarak döndürülebilmektedir. Describethis internet üzerinde Dublin Core standardının kullanımının artmasını ve bu standardın geliştirilmesini desteklemektedir.

DCS aşağıdaki formatları desteklemektedir:

- HTML ve XHTML Dokümanları
- Dublin Core/RDF
- Dublin Core/XML
- Dublin Core/HTML (META takıları)
- GIF, JPG (EXIF) ve diğer resim formatları
- RSS
- bibTex

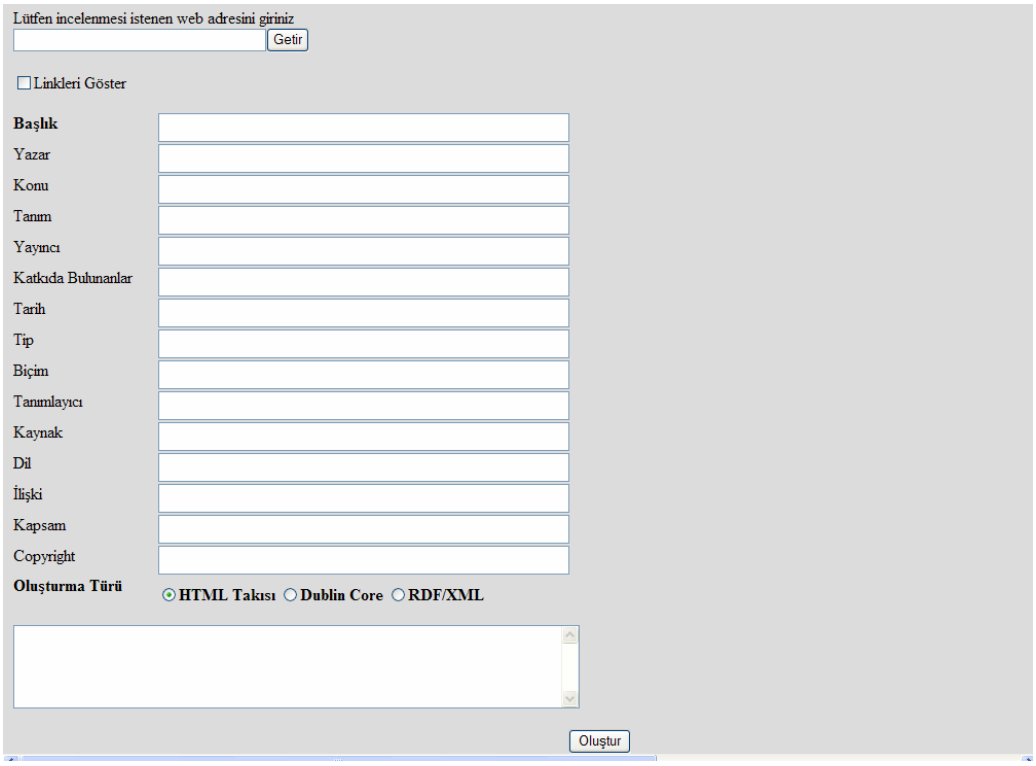
4. TASARIM VE GERÇEKLEŞTİRME

Bu çalışmada Türkçe dil destekli otomatik bir metadata üretici programı tasarlanıp geliştirilmiştir. Metadata üretmeyi sadece bilgisayar kullanılarak yapabilmenin yanında uzman kişiler kullanarak da metadata oluşturmak mümkündür. Ancak daha önce belirtildiği gibi metadata üretiminde en uygun çözüm insan ve bilgisayar kullanımının bir arada sunulduğu çözümdür. Bilgisayarların hızlı işlem yapma gücüne insan zekâsı ve yorumlama gücü eklenerek çok daha iyi sonuçlar alabilmek mümkündür. Bu sebeple oluşturulan programda insan ve bilgisayar odaklı bir yaklaşım benimsenmiştir.

Oluşturulan programın bazı kısıtları mevcuttur. Metadata üretici programı, sayfanın kaynak kodu üzerinde işlem yaparken HTML takılarını başlangıç ve bitiş noktası olarak belirlediği için iyi yapılandırılmış HTML sayfaları üzerinde çalışmaktadır. Türkçe karakter desteğinin sağlanabilmesi için programın yüklü bulunduğu sunucunun Türkçe karakterleri destekleyebilmesi gerekmektedir. Program, Windows XP Türkçe işletim sistemine sahip, P4 2,6 Ghz işlemci, 1024 MB RAM'e sahip bir bilgisayarda denenmiştir.

Oluşturulan program Microsoft Visual Studio 2005 üzerinde C# ASP.NET dili kullanılarak gerçekleştirilmiştir. Visual Studio programının kullanılma sebebi kullanıcı dostu bir ara yüze sahip olması ve sağladığı hazır sınıflarla program geliştirmede kolaylık sağlayabilmesidir.

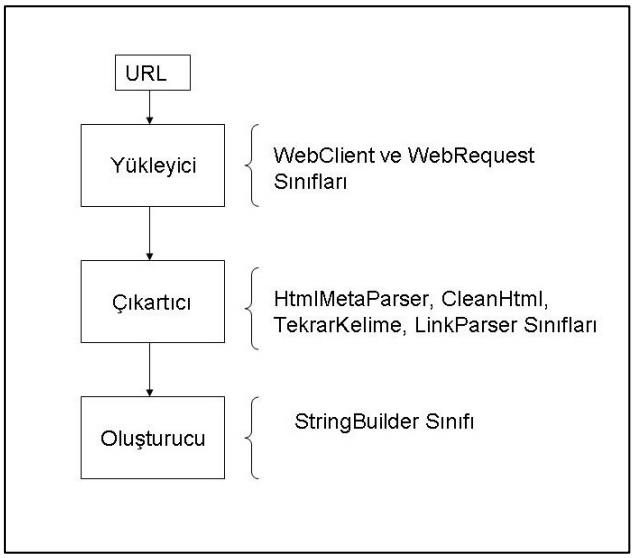
Metadata üreticisi oluşturmak için çalışmamızda basit bir kullanıcı ara yüzü kullanılmıştır. Bu ara yüz Şekil 5'te gösterilmektedir.



Şekil 4.1: Kullanıcı Ara Yüzü

Kullanıcı ara yüzünde on altı adet metin kutusu bulunmaktadır. Kullanıcı en üstte bulunan metin kutusuna metadata üretmek istediği sitenin URL'sini başında "http" olacak şekilde girip getir tuşuna basmaktadır. Bu işlemden sonra girilen URL'deki sayfanın kaynak kodundan gerekli kısımlar alınarak ekrana ve metin kutularına yazılmaktadır.

Program Yükleyici, Ayrıştırıcı ve Oluşturucu olmak üzere üç bölümlü olarak tasarlanmıştır. Yükleyici kısmı ile metadata'sı oluşturulmak istenen kaynağın URL'si alınarak kaynak kodu indirilmektedir. Ayrıştırıcı kısmında yükleyici ile indirilen kaynak kodunda bulunan bazı özel takılar (başlık, koyu harf, link gibi) arasında kalan kısımlar çıkarılmaktadır. En son Oluşturucu bölümü ile de indirilen kaynak kodundan çıkarılan bilgilerle Dublin Core/XML, Dublin Core/RDF ya da Dublin Core /Metadata şeklinde metadata oluşturulmaktadır. Şekil 6'da programın çalışma şeması gösterilmektedir.



Şekil 4.2: Programın Çalışma Şeması

Programda dört adet sınıf kullanılmıştır. Bu sınıflar CleanHtml, HtmlMetaParser, LinkParser ve TekrarKelime sınıflarıdır.

4.1 HtmlMetaParser Sınıfı

Bu sınıf, boldBul, titleBul ve Parse olmak üzere 3 ana metottan oluşmaktadır. Bu metotlardan boldBul, girilen URL'deki kaynak kodu tarayarak o kaynak kod içerisinde " " takıları arasında kalan koyu yazılmış yani vurgulanmış metinleri bulmaktadır. Metin içerisinde arama yapabilmek için Düzenli İfadelerden (*Regular Expressions*) yararlanılmıştır. Düzenli ifadeler bir metni düzenlemek ya da metnin içinden belirli kurallara uyan alt metinleri seçmemize yarayan bir dildir. Düzenli ifadeler birçok programlama dili ve metin editöründe belirli örneklere uyan metinleri bulmak ve üzerlerinde işlem yapabilmek için kullanılmakta ve *String* türünde karakter topluluklarına uygulanmaktadır. Düzenli ifadelerde öncelikle bir desen (*Pattern*) oluşturmak gerekmektedir. C# dilinde, oluşturulan bu desene uyan ifadeler bir dizi ile ulaşmak mümkündür.
titleBul metodu, sayfanın başlık bilgisini HTML kaynak kodundan düzenli ifadelerle almakta ve ilgili metin kutusuna yazmaktadır.

Parse metodu HtmlMetaParser sınıfının en önemli metodudur. Bu metot ile HTML kaynak kodunda bulunan “Meta” takıları düzenli ifadeler yardımı ile bulunmaktadır. Meta takılarından elde edilen içerik bilgileri daha sonra Dublin Core standardında oluşturulmak üzere metin kutularına aktarılmaktadır. Şekil 9’da gösterilen kodlar ile bu işlem iki aşamalı olarak yapılmaktadır. İlk olarak sadece sayfanın başında bulunan “Meta” takılarını içerikleri ile bulup almak için bir düzenli ifade oluşturulmuştur. Şekil 9’da gösterilen koddaki ilk düzenli ifade “<meta” ile başlayan takıları seçmeyi sağlamaktadır. Oluşturulan jenerik MetaList listesi düzenli ifadeler sonucu bulunan Meta takılarını eklemek için oluşturulmuştur. Jenerik sınıfı C# 2.0 ile gelen bir yeniliktir. Bu sınıfın sunduğu “*List*” sınıfı sayesinde yapı olarak *ArrayList*’e benzeyen ancak içine sadece belirlenen veri türünün eklenebileceği listeler oluşturmak mümkündür. HtmlMetaParser sınıfında MetaList jenerik listesinin *HtmlMeta* türünden veri alması sağlanmaktadır. *HtmlMeta* sınıfı HTML “Meta” takılarına program yoluyla erişime izin veren bir sınıftır. Bu sınıfın “*Name*” ve “*Content*” isminde iki özelliği (*Property*) bulunmaktadır. “Name” ve “Content” özelliğine değerler atayarak “Meta” takıları oluşturabilmek mümkündür. “Name” özelliği oluşturulan “Meta” takısının ismini “Content” özelliği ise içeriğini vermektedir. Bu sınıf sayesinde ikinci gruptaki düzenli ifadelerle Meta takılarının İsim ve İçerik özellikleri MetaList listesine eklenebilmektedir. Şekil 7’de HtmlMetaParser sınıfının metotları gösterilmektedir.

```
    public static void boldBul(string kaynakKod) {

Regex boldBulucu = new Regex(@"(<b>)(?<yazi>(.*))(</b>)", RegexOptions.IgnoreCase |
    RegexOptions.ExplicitCapture);

System.Web.HttpContext.Current.Response.Write("BOLD YAZILMIS KELIMELER" + "</br>");
foreach (Match tut in boldBulucu.Matches(kaynakKod)) {
    System.Web.HttpContext.Current.Response.Write(tut.Value.ToString() + " ");
                }
    }
```

Şekil 4.3: boldBul Metodu

```
public static string titleBul(string kaynak)
   {

       Regex titleBulucu = new Regex(@"(<title>)(?<yazi>(.*))(</title>)", RegexOptions.IgnoreCase |
RegexOptions.ExplicitCapture);
       string basi="";
       foreach (Match baslik in titleBulucu.Matches(kaynak))
       {
          basi = baslik.Groups["yazi"].Value.ToString();

       }

       return basi;

   }
```

Şekil 4.4: titleBul Metodu

```
public static List<HtmlMeta> Parse(string htmldata)
 {
    Regex metaregex =
       new Regex(@"<meta\s*(?:(?:\b(\w|-)+\b\s*(?:=\s*(?:""[^""]*""|'" +
              @"[^']*'|[^""'<> ]+)\s*)?)*)/?\s*>",
              RegexOptions.IgnoreCase | RegexOptions.ExplicitCapture);

    List<HtmlMeta> MetaList = new List<HtmlMeta>();

    foreach (Match metamatch in metaregex.Matches(htmldata))
    {
       HtmlMeta mymeta = new HtmlMeta();

       Regex submetaregex =
          new Regex(@"(?<name>\b(\w|-)+\b)\" +
                @"s*=\s*(""(?<value>" +
                @"[^""]*)""|'(?<value>[^']*)'" +
                @"|(?<value>[^""'<> ]+)\s*)+",
                RegexOptions.IgnoreCase |
                RegexOptions.ExplicitCapture);

       foreach (Match submetamatch in
             submetaregex.Matches(metamatch.Value.ToString()))
       {
          if ("http-equiv" ==
               submetamatch.Groups["name"].ToString().ToLower())
             mymeta.HttpEquiv =
               submetamatch.Groups["value"].ToString();

          if (("name" ==
               submetamatch.Groups["name"].ToString().ToLower())
               && (mymeta.HttpEquiv == String.Empty))
             mymeta.Name = submetamatch.Groups["value"].ToString();

          if ("scheme" ==
             submetamatch.Groups["name"].ToString().ToLower())
             mymeta.Scheme = submetamatch.Groups["value"].ToString();

          if ("content" ==
             submetamatch.Groups["name"].ToString().ToLower())
          {
             mymeta.Content = submetamatch.Groups["value"].ToString();
             MetaList.Add(mymeta);
          }
       }
    }

     return MetaList;
 }
```

Şekil 4.5: Parse Metodu

4.2 CleanHtml Sınıfı

Bu sınıf 2 metottan oluşmaktadır. Şekil 10'da gösterilen Clean metodu, *String* türünden bir parametre olarak aldığı HTML kaynak kodunda bulunan HTML takılarını atmaktadır. Bu işlemi için de düzenli ifadeler kullanılmaktadır. *Regex.Replace* metodu düzenli ifadelerde arama ve değiştirme işlemini yapmaktadır. Bu metot üç parametre alır. Birinci parametre üzerinde işlem yapılan *String* türünden ifadeyi, ikinci parametre değiştirilecek örnekleri ve üçüncü parametre de ne ile değiştirileceğini belirtmektedir. Clean metodunda amaç alınan kaynak kodda bulunan HTML takılarını atmak olduğu için bu metotta atılması gereken ifadeler boşluk ile değiştirilmiştir. Bu metot sonunda parametre olarak alınan kaynak kodunun "Body" kısmını HTML takıları atılmış olarak kelime kelime geri döndürmektedir.

stringBol metodu, Clean metodu ile HTML takılarından temizlenmiş olan kaynak kodunun döndürdüğü kelimeleri daha sonra kullanmak için jenerik bir listeye atamaktadır. Bu metodun döndürdüğü jenerik liste daha sonra tekrar eden kelimeleri bulmak için kullanılmaktadır. Şekil 11'de stringBol metodu gösterilmektedir.

```
public static string Clean(string Contents)
      {
         if (Contents.IndexOf("startnav") != -1)
         {
            int startnav = Contents.IndexOf("startnav");
            int endnav = Contents.IndexOf("endnav");
            int lengthnav = endnav - startnav;
            string subContents = Contents.Substring(startnav, lengthnav);
            Contents = Contents.Replace(subContents, "");
         }
         Contents = Regex.Replace(Contents,
"<(select|option|script|style|title)(.*?)>((.|\n)*?)</(select|option|script|style|title)>", " ",
RegexOptions.IgnoreCase);
         Contents = Regex.Replace(Contents, "&(nbsp|quot|copy|bull);", "");
         Contents = Regex.Replace(Contents, "&amp;", "&");
         Contents = Regex.Replace(Contents, "<[^>]*>", "");
         Contents = Regex.Replace(Contents, "<([\\s\\S])+?>", " ",
RegexOptions.IgnoreCase).Replace("  ", " ");
         Contents = Regex.Replace(Contents, "<[^<>]+>", " ", RegexOptions.IgnoreCase);
         //Contents = Regex.Replace("(<(\w+)[^>]*?>(.*?)</\1>", "$1");
         Contents = Regex.Replace(Contents, "\\W", " ");
         //Trace.Warn("File Contents", Contents);
         return Contents;
      }
```

Şekil 4.6: Clean Metodu

```
public static List<string> strinbBol(string htmlkaynak) {

         List<string> bolunmusKaynak=new List<string>();
         string s = htmlkaynak;
         //char[] seps = new char[] { ' ' };
         Regex o = new Regex(@"[\s]+");
         foreach (string ss in o.Split(htmlkaynak))
         {

            bolunmusKaynak.Add(ss);

         }

         return bolunmusKaynak;

      }
```

Şekil 4.7: stringBol Metodu

4.3 TekrarKelime Sınıfı

Bu sınıf KelimeBulucu isimli bir metoda sahiptir. KelimeBulucu metodu *String* dizisi türünde bir parametre almaktadır. *String* dizisinin içeriği tekrar eden eleman olmayacak şekilde bir ArrayList'e aktarılmaktadır. Eleman tekrar sayısını tutmak için yeni bir dizi oluşturulduktan sonra iç içe For döngüsü ile tekrarsız dizideki elemanlar asıl dizi ile karşılaştırılmaktadır. Sonuç olarak iki dizi elde edilmekte birinci dizide kelimeler bulunurken ikinci dizide o kelimenin indeks numarasına sahip yerde tekrar sayısı yazılmaktadır. Metot sonunda tekrar sayısı 5'ten fazla olan kelimeler ekrana yazılmaktadır. Şekil 12'de KelimeBulucu metodu gösterilmektedir.

```
public static void KelimeBulucu(string[] kelimeDizisi)
   {
      ArrayList diziElemanlari = new ArrayList();

            for (int i = 0; i < kelimeDizisi.Length; i++)
      {
         if (diziElemanlari.Contains(kelimeDizisi[i]) == false)
         {
            diziElemanlari.Add(kelimeDizisi[i]);
         }
      }
      int[] elemanTekrarlari = new int[diziElemanlari.Count];
      for (int i = 0; i < diziElemanlari.Count; i++)
      {
         elemanTekrarlari[i] = 0;
         string siradaki = diziElemanlari[i].ToString();
         for (int j = 0; j < kelimeDizisi.Length; j++)
         {
            if (siradaki == kelimeDizisi[j])
            {
               elemanTekrarlari[i]++;
            }
         }
      }
     for (int i = 0; i < diziElemanlari.Count; i++)
      {
         if(elemanTekrarlari[i]>5)
      System.Web.HttpContext.Current.Response.Write("Kelime : " + diziElemanlari[i].ToString() + " -
Tekrar Sayısı : " + elemanTekrarlari[i].ToString() + "</br>");

      }
   }
```

Şekil 4.8: KelimeBulucu metodu

4.4 LinkParser Sınıfı

Bu sınıf parseLink isimli bir metottan oluşmaktadır. Bu metot girilen URL'deki sayfanın kaynak kodunu string parametresi olarak almaktadır. Girilen sayfada bulunan linkleri tutacak iki adet ArrayList oluşturularak düzenli ifadelerle elde edilen linkler bu ArrayList'lere eklenmektedir. Bu metotta iki adet düzenli ifade kullanılmıştır. Birinci düzenli ifade HTML'de metinlere link vermeye yarayan "<A>" takısı ile resimlere link vermeye yarayan "<AREA>" takılarını bulmaya çalışmaktadır. Her iki takıda da bağlantı verilen sayfa "HREF" özelliği ile belirtildiği için ikinci düzenli ifade de elde edilen linklerde "HREF" özelliği kontrol edilmekte ve bunun değeri "link" değişkenine atanmaktadır. Linkleri belirlerken önemli olan nokta sayfa içine veya dışına verildiklerini anlamaktır. Bu sebeple oluşturduğumuz ArrayList'lere eklemeden önce String türünden oluşturulan "link" değişkeni kontrol edilmektedir. Elde edilen linkleri sayfada düzenli gösterebilmek adına bir StringBuilder sınıfının bir örneği oluşturulmuştur. StringBuilder'in normal String sınıfına göre üstünlüğü birleştirme (Concat) işlemlerinde StringBuilder'ın 50 ile 100 kat arasında daha iyi bir performans göstermesidir. StringBuilder sınıfı ile oluşturulan string metot sonunda döndürülmektedir. Şekil 13'de parseLink metodu gösterilmektedir.

```
public static string ParseLinks(string htmlData)
{
StringBuilder str = new StringBuilder();
ArrayList linkLocal = new ArrayList();
ArrayList linkExternal = new ArrayList();
string link = String.Empty;
foreach (Match match in Regex.Matches(htmlData
, @"(?<anchor><\s*(a|area)\s*(?:(?:\b\w+\b\s*(?:=\s*(?:""[^""]*""|'[^']*'|[^""'<> ]+)\s*)?)*)?\s*>)"
, RegexOptions.IgnoreCase | RegexOptions.ExplicitCapture))
{
link = String.Empty;
foreach (Match submatch in Regex.Matches(match.Value.ToString()
, @"(?<name>\b\w+\b)\s*=\s*(""(?<value>[^""]*)""|'(?<value>[^']*)'|(?<value>[^""'<> \s]+)\s*)+"
, RegexOptions.IgnoreCase | RegexOptions.ExplicitCapture))
{
if ("href" == submatch.Groups[1].ToString().ToLower())
{
link = submatch.Groups[2].ToString();
break;
}
}
if (link.IndexOf("#") > -1)
{
link = link.Substring(0, link.IndexOf("#"));
 }
    if (link.IndexOf("javascript:") == -1
 & link.IndexOf("mailto:") == -1
&& !link.StartsWith("#")
&& link != String.Empty)
   {
 f ((link.Length > 8) && (link.StartsWith("http://") || link.StartsWith("https://") ||
link.StartsWith("file://")|| link.StartsWith("//")|| link.StartsWith(@"\\")))

            {
               linkExternal.Add(link);
            }
            else if (link.StartsWith("?"))
            {

               linkLocal.Add(link);
            }
            else
            {
               linkLocal.Add(link);
            }
         }
      }
str.Append("<table cellpadding=\"0\" cellspacing=\"0\" border=\"1px\" width=\"100%\"><tr><td>
Site İçi Linkler</td></tr>");
 foreach (string lnk in linkLocal.ToArray(typeof(string)))
 str.Append(String.Format("<tr><td>{0}</td></tr>", lnk));
 str.Append("</table><br/><br/>");
str.Append("<table cellpadding=\"0\" cellspacing=\"0\" border=\"1px\" width=\"100%\"><tr><td>Site
Dışı Linkler</td></tr>");
```

Şekil 4.9: ParseLinks Metodu

4.5 Default.aspx.cs Sınıfı

Bu sınıf, programın asıl çalışma mantığını içeren sınıftır. ASP.NET ile ortaya çıkan Geri Kod (*Code Behind*) ile tasarım yüzü ile asıl programlamanın olduğu bölüm birbirinden ayrılmıştır. Bu sayede işaretleme kodları ile programlama kodlarının oluşturduğu karmaşa ortadan kalkmıştır.

Oluşturulan programda, girilen URL'deki sayfanın kaynak kodunu almak için WebClient ve WebRequest sınıfları kullanılmıştır. Kaynak kodu almak için iki sınıf kullanmanın sebebi yapılan denemelerde WebClient sınıfının Türkçe karakterlerin bulunduğu sayfalarda doğru çalışmamasıdır. Bu nedenle Türkçe sayfaların kaynak kodunu getirmek için WebRequest sınıfı kullanılmıştır. Program girilen URL'nin uzantısını düzenli ifadeler ile kontrol etmekte ve eğer "tr" uzantısına sahip bir URL girilmişse bu sayfanın kaynak kodunu WebRequest sınıfı ile getirmektedir. WebRequest sınıfı, HtmlMetaParser sınıfının GetHttpPage metodunda kullanılmıştır. Bu metot sayfanın URL adresini parametre olarak almakta ve sayfanın kaynak kodunu StreamReader sınıfı ile okumaktadır. Türkçe sayfaların kaynak kodu okunurken karakter kodlaması için Türkçe için kullanılan "Windows–1254" kodlaması (*Encoding*) kullanılmıştır. Şekil 14'de GetHttpPage metodu gösterilmektedir.

"Tr" uzantısı olmayan sayfalar için WebClient sınıfı kullanılmıştır. Bu sınıfın DownloadData metoduna verilen URL parametresi ile istenen sayfanın kaynak kodu indirilmektedir.

Programda oluşturulan XML dosyaları, default.aspx.cs sayfasında oluşturulmaktadır. C# dilinin XmlTextWriter sınıfı ile kolaylıkla XML dosyası oluşturmak mümkündür. Dublin Core metadata standardında kullanıcı bütün elementleri girmek zorunda olmadığından metadata'nın sadece dolu olan metin kutularından oluşturulması gerekmektedir. Bu sebeple oluşturma aşamasında metin kutularının içeriği kontrol edilmekte ve boş olan metin kutusu "GOTO" ifadesi

ile atlanarak bir sonraki metin kutusunun içeriği kontrol edilmektedir. Şekil 15'de XMLTextWriter sınıfı ve kullanımı gösterilmektedir.

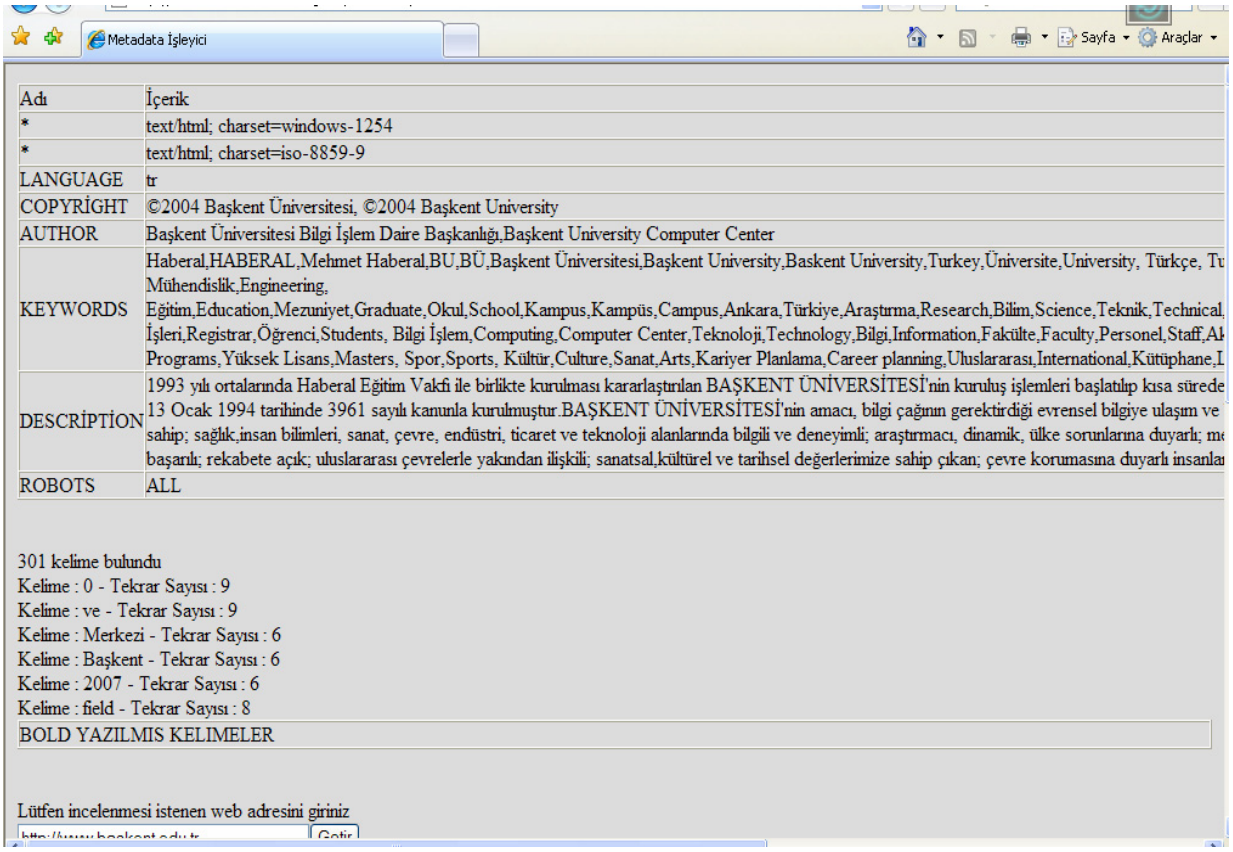
```
public static string GetHttpPage(string pageUrl)
   {
      System.Net.WebResponse response = null;
      string strSayfa;
      try
      {
         System.Net.WebRequest request =
          System.Net.WebRequest.Create(pageUrl);
         request.Timeout = 10000;

         response = request.GetResponse();

         System.IO.Stream streamReceive = response.GetResponseStream();
         System.Text.Encoding encoding = System.Text.Encoding.GetEncoding("windows-1254");
         System.IO.StreamReader streamRead = new System.IO.StreamReader(streamReceive,
encoding);
         strSayfa = streamRead.ReadToEnd();
         return strSayfa;
      }
```

Şekil 4.10: GetHttpPage Metodu

```
        XmlTextWriter XmlYaz = new XmlTextWriter(Server.MapPath("dublinCore.xml"),
Encoding.UTF8);
            XmlYaz.WriteStartDocument();
            XmlYaz.Formatting = Formatting.Indented;
            XmlYaz.Indentation = 4;
            XmlYaz.WriteStartElement("record");
            XmlYaz.WriteAttributeString("xmlns:dc", "http://purl.org/dc/elements/1.1");
            if (txtTitle.Text == "") goto zero;
            XmlYaz.WriteElementString("dc:title", this.txtTitle.Text);
        zero:
            if (txtYazar.Text == "") goto Bir;
            XmlYaz.WriteElementString("dc:creator", this.txtYazar.Text);
        Bir:
            if (txtKonu.Text == "") goto Iki;
            XmlYaz.WriteElementString("dc:subject", this.txtKonu.Text);
        Iki:
            if (txtTanim.Text == "") goto Uc;
            XmlYaz.WriteElementString("dc:description", this.txtTanim.Text);
        Uc:
            if (txtYayinci.Text == "") goto Dort;
            XmlYaz.WriteElementString("dc:publisher", this.txtYayinci.Text);
        Dort:
            if (txtContributor.Text == "") goto Bes;
            XmlYaz.WriteElementString("dc:contributor", this.txtContributor.Text);
        Bes:
            if (txtTarih.Text == "") goto Alti;
            XmlYaz.WriteElementString("dc:date", this.txtTarih.Text);
        Alti:
            if (txtTip.Text == "") goto Yedi;
            XmlYaz.WriteElementString("dc:type", this.txtTip.Text);
        Yedi:
            if (txtBicim.Text == "") goto Sekiz;
            XmlYaz.WriteElementString("dc:format", this.txtBicim.Text);
        Sekiz:
            if (txtTanimlayici.Text == "") goto Dokuz;
            XmlYaz.WriteElementString("dc:identifier", this.txtTanimlayici.Text);
        Dokuz:
            if (txtKaynak.Text == "") goto On;
            XmlYaz.WriteElementString("dc:source", this.txtKaynak.Text);
        On:
            if (txtDil.Text == "") goto Onbir;
            XmlYaz.WriteElementString("dc:language", this.txtDil.Text);
        Onbir:
            if (txtIliski.Text == "") goto Oniki;
            XmlYaz.WriteElementString("dc:relation", this.txtIliski.Text);
        Oniki:
            if (txtKapsam.Text == "") goto Onuc;
            XmlYaz.WriteElementString("dc:coverage", this.txtKapsam.Text);
        Onuc:
            if (txtCopyright.Text == "") goto Ondort;
            XmlYaz.WriteElementString("dc:rights", this.txtCopyright.Text);
        Ondort:
            XmlYaz.WriteEndElement();
            XmlYaz.WriteEndDocument();
            XmlYaz.Close();
```

Şekil 4.11: XMLTextWriter Kullanımı

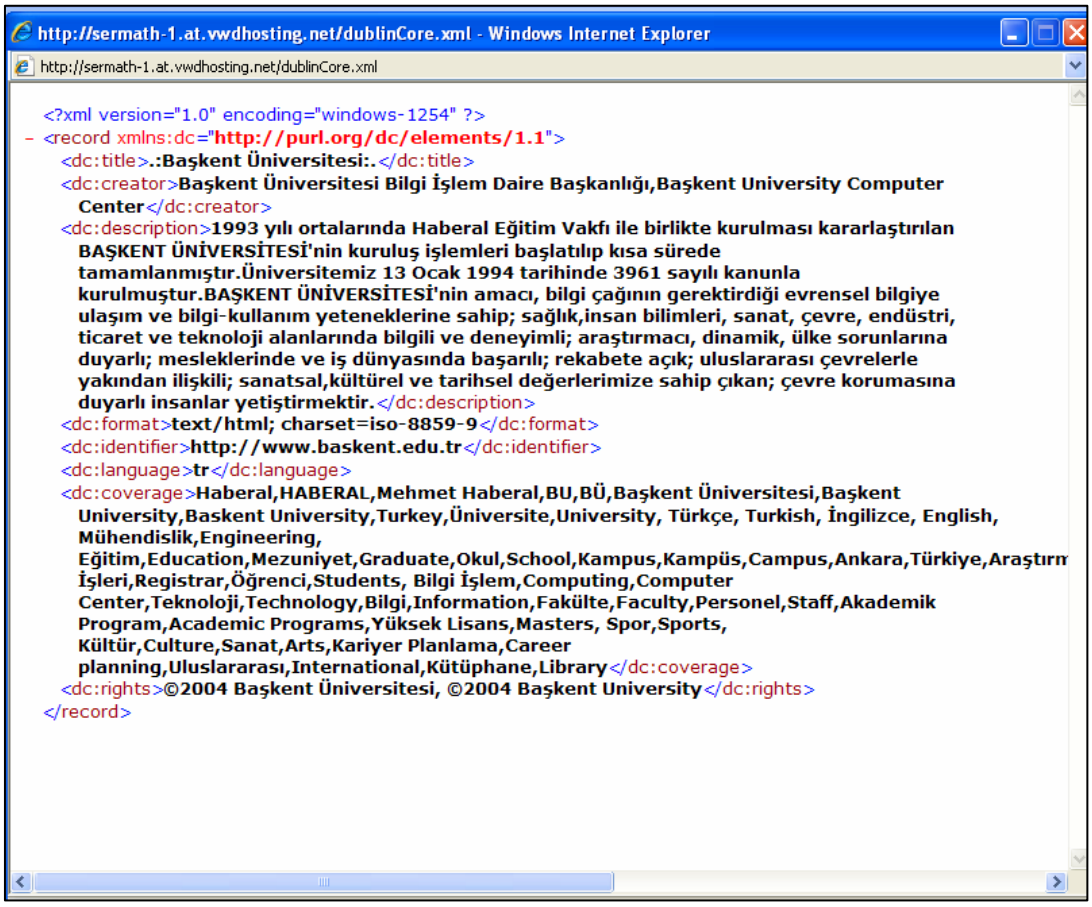
Oluşturulan program Türkçe Web sayfalarında denenmiş ve piyasade bulunan Dc.dot, DISA gibi Dublin Core metadata üreticilerinin aksine Türkçe karakter desteği sağlamıştır. Programın www.baskent.edu.tr sitesinde denenmiş görüntüleri Şekil 16 ve 17'de gösterilmektedir.



Şekil 4.12: Başkent Üniversitesi'nin sayfasından alınan metadata'lar

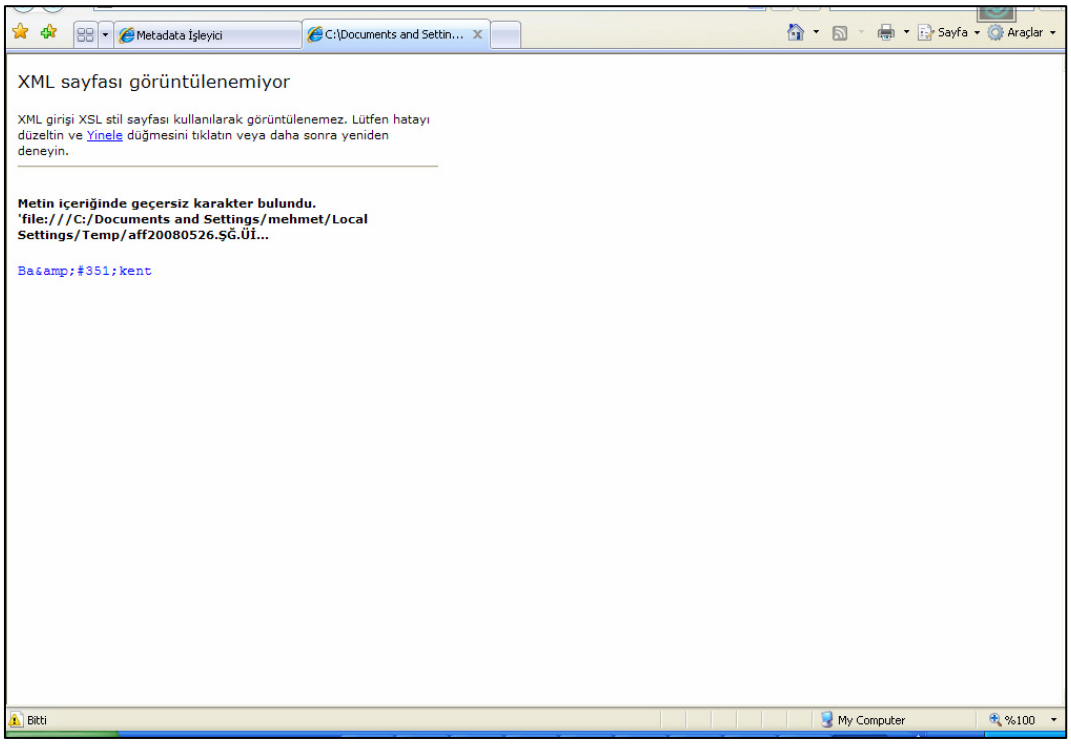
```
<meta name="DC.Title"  content=".:Başkent Üniversitesi:.">
<meta name="DC.Creator" content="Başkent Üniversitesi Bilgi İşlem Daire Başkanlığı,Başkent
University Computer Center">
<meta name="DC.Format" content="text/html; charset=iso-8859-9">
<meta name="DC.Identifier" content="http://www.baskent.edu.tr">
<meta name="DC.Language" content="tr">
<meta name="DC.Coverage" content="Haberal,HABERAL,Mehmet Haberal,BU,BÜ,Başkent
Üniversitesi,Başkent University,Baskent University,Turkey,Üniversite,University, Türkçe,
Turkish, İngilizce, English, Mühendislik, Engineering, Eğitim, Education, Mezuniyet ,Graduate,
Okul
,School,Kampus,Kampüs,Campus,Ankara,Türkiye,Araştırma,Research,Bilim,Science,Teknik,Te
chnical,Akademik,Academics,Öğrenci İşleri,Registrar,Öğrenci,Students, Bilgi İşlem, Computing,
Computer
Center,Teknoloji,Technology,Bilgi,Information,Fakülte,Faculty,Personel,Staff,Akademik
Program,Academic Programs,Yüksek Lisans,Masters, Spor,Sports, Kültür, Culture, Sanat, Arts,
Kariyer Planlama,Career planning,Uluslararası,International,Kütüphane,Library">
<meta name="DC.Rights" content="©2004 Başkent Üniversitesi, ©2004 Başkent University ">
```

Şekil 4.13: Başkent Üniversitesi'nin sayfasından alınan bilgilerle oluşturulan DC Metadata



Şekil 4.14: Başkent Üniversitesi'nin sayfasından alınan bilgilerle oluşturulan DC/XML Metadata

DISA Dublin Core Metadata Üreticisi ile Türkçe karakterler kullanılarak metadata üretmeye çalışıldığında Şekil 18'de görülen hata ile karşılaşılmıştır.



Şekil 4.15: DISA ile Türkçe karakter içeren DC Metadata Oluşturmak

Aynı şekilde DC.dot metadata üreticisini kullanarak Başkent Üniversitesi'nin sayfasından metadata oluşturmak istediğimizde Şekil 20'de görülen Türkçe karakterlerin bozuk olduğu çıktıyı almaktayız.

DCdot

Dublin Core metadata editor

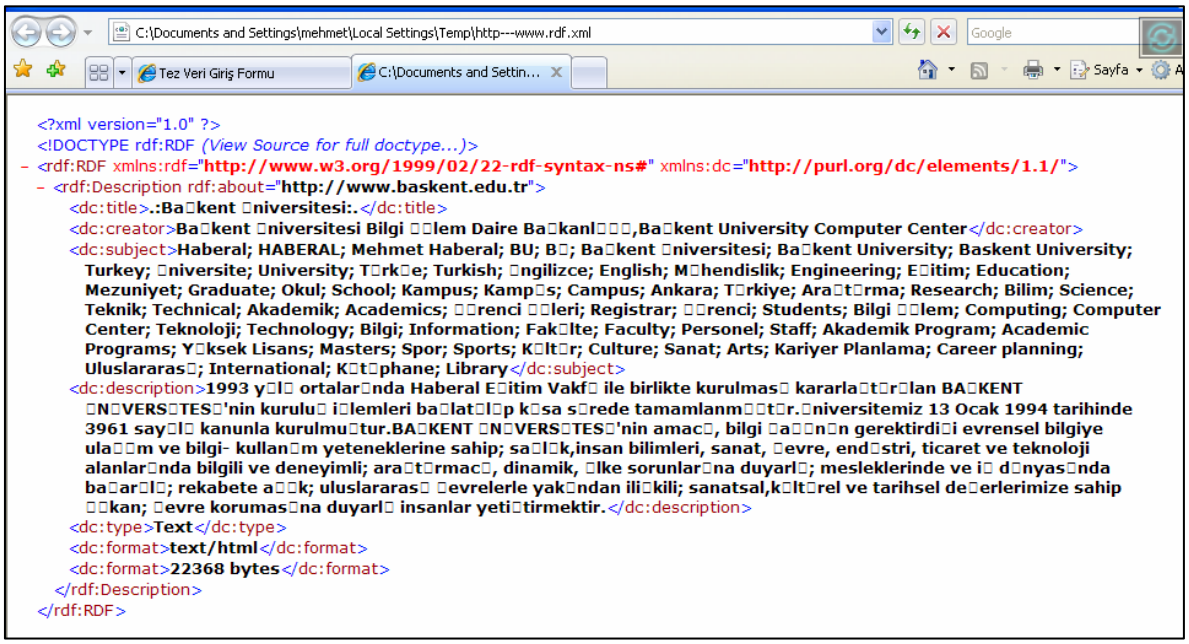
UKOLN

Results for **URL**: http://www.baskent.edu.tr [summary]

```
<link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
<link rel="schema.DCTERMS" href="http://purl.org/dc/terms/" />
<meta name="DC.title" content=".:Ba?kent ?niversitesi:." />
<meta name="DC.creator" content="Ba?kent ?niversitesi Bilgi ??lem Daire Ba?kanl???,Ba?kent University Computer Center" />
<meta name="DC.subject" content="Haberal; HABERAL; Mehmet Haberal; BU; B?; Ba?kent ?niversitesi; Ba?kent University;
Baskent University; Turkey; ?niversite; University; T?rk?e; Turkish; ?ngilizce; English; M?hendislik; Engineering; E?itim;
Education; Mezuniyet; Graduate; Okul; School; Kampus; Kamp?s; Campus; Ankara; T?rkiye; Ara?t?rma; Research; Bilim; Science;
Teknik; Technical; Akademik; Academics; ??renci ??leri; Registrar; ??renci; Students; Bilgi ??lem; Computing; Computer
Center; Teknoloji; Technology; Bilgi; Information; Fak?lte; Faculty; Personel; Staff; Akademik Program; Academic Programs;
Y?ksek Lisans; Masters; Spor; Sports; K?lt?r; Culture; Sanat; Arts; Kariyer Planlama; Career planning; Uluslararas?;
International; K?t?phane; Library" />
<meta name="DC.description" content="1993 y?l? ortalar?nda Haberal E?itim Vakf? ile birlikte kurulmas? kararla?t?r?lan
BA?KENT ?N?VERS?TES?'nin kurulu? i?lemleri ba?lat?l?p k?sa s?rede tamamlanm??t?r.?niversitemiz 13 Ocak 1994 tarihinde 3961
say?l? kanunla kurulmu?tur.BA?KENT ?N?VERS?TES?'nin amac?, bilgi ?a??n?n gerektirdi?i evrensel bilgiye ula??m ve
bilgi-kullan?m yeteneklerine sahip; sa?l?k,insan bilimleri, sanat, ?evre, end?stri, ticaret ve teknoloji alanlar?nda
bilgili ve deneyimli; ara?t?rmac?, dinamik, ?lke sorunlar?na duyarl?; mesleklerinde ve i? d?nyas?nda ba?ar?l?; rekabete
a??k; uluslararas? ?evrelerle yak?ndan ili?kili; sanatsal,k?lt?rel ve tarihsel de?erlerimize sahip ??kan; ?evre korumas?na
duyarl? insanlar yeti?tirmektir." />
<meta name="DC.type" scheme="DCTERMS.DCMIType" content="Text" />
<meta name="DC.format" content="text/html" />
<meta name="DC.format" content="22368 bytes" />
<meta name="DC.identifier" scheme="DCTERMS.URI" content="http://www.baskent.edu.tr" />
```

[how to use the XHTML description]

Şekil 4.16: DC.dot ile Başkent Üniversitesi'nin sayfasından Metadata oluşturmak



C:\Documents and Settings\mehmet\Local Settings\Temp\http---www.rdf.xml

Tez Veri Giriş Formu | C:\Documents and Settin...

```
<?xml version="1.0" ?>
<!DOCTYPE rdf:RDF (View Source for full doctype...)>
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/dc/elements/1.1/">
  - <rdf:Description rdf:about="http://www.baskent.edu.tr">
      <dc:title>.:Ba□kent □niversitesi:.</dc:title>
      <dc:creator>Ba□kent □niversitesi Bilgi □□lem Daire Ba□kanl□□□,Ba□kent University Computer Center</dc:creator>
      <dc:subject>Haberal; HABERAL; Mehmet Haberal; BU; B□; Ba□kent □niversitesi; Ba□kent University; Baskent University;
        Turkey; □niversite; University; T□rk□e; Turkish; □ngilizce; English; M□hendislik; Engineering; E□itim; Education;
        Mezuniyet; Graduate; Okul; School; Kampus; Kamp□s; Campus; Ankara; T□rkiye; Ara□t□rma; Research; Bilim; Science;
        Teknik; Technical; Akademik; Academics; □□renci □□leri; Registrar; □□renci; Students; Bilgi □□lem; Computing; Computer
        Center; Teknoloji; Technology; Bilgi; Information; Fak□lte; Faculty; Personel; Staff; Akademik Program; Academic
        Programs; Y□ksek Lisans; Masters; Spor; Sports; K□lt□r; Culture; Sanat; Arts; Kariyer Planlama; Career planning;
        Uluslararas□; International; K□t□phane; Library</dc:subject>
      <dc:description>1993 y□l□ ortalar□nda Haberal E□itim Vakf□ ile birlikte kurulmas□ kararla□t□r□lan BA□KENT
        □N□VERS□TES□'nin kurulu□ i□lemleri ba□lat□l□p k□sa s□rede tamamlanm□□t□r.□niversitemiz 13 Ocak 1994 tarihinde
        3961 say□l□ kanunla kurulmu□tur.BA□KENT □N□VERS□TES□'nin amac□, bilgi □a□□n□n gerektirdi□i evrensel bilgiye
        ula□□m ve bilgi- kullan□m yeteneklerine sahip; sa□l□k,insan bilimleri, sanat, □evre, end□stri, ticaret ve teknoloji
        alanlar□nda bilgili ve deneyimli; ara□t□rmac□, dinamik, □lke sorunlar□na duyarl□; mesleklerinde ve i□ d□nyas□nda
        ba□ar□l□; rekabete a□□k; uluslararas□ □evrelerle yak□ndan ili□kili; sanatsal,k□lt□rel ve tarihsel de□erlerimize sahip
        □□kan; □evre korumas□na duyarl□ insanlar yeti□tirmektir.</dc:description>
      <dc:type>Text</dc:type>
      <dc:format>text/html</dc:format>
      <dc:format>22368 bytes</dc:format>
    </rdf:Description>
  </rdf:RDF>
```

Şekil 4.17:Dc.dot ile Oluşturulan DC/RDF

5. SONUÇ VE ÖNERİLER

Web üzerinde bulunan elektronik verilere erişim gün geçtikçe zorlaşmaktadır. Bu zorluğun üstesinden gelmemizi sağlayacak etkin yöntemler geliştirilmezse kısa bir süre sonra Web üzerinde aradığımız veriye ulaşmamız mümkün olmayacaktır. Bu açıdan metadata kullanımı ve kullanılan metadatalarda standardizasyon büyük önem taşımaktadır.

Yapılan araştırmalarda piyasada bulunan metadata üretici programlarının çoğunun Türkçe karakterleri doğru göstermediği görülmüştür. Oluşturulan programın diğer programlara göre üstün tarafı kullanıcı ara yüzünün tamamen Türkçe olması ve Türkçe karakterleri desteklemesidir.

Oluşturulan metadata üreticisi, HTML dilinden hiç anlamayan bir kullanıcının bile sadece URL adresini girip bir düğmeye basarak HTML kaynak kodunda yazılmış bulunan Title, Description, Keywords gibi metadata takılarında bulunan bilgiler ve sayfa içerisinde tekrarlanan kelimeleri kullanarak Dublin Core metadata element kümesine uygun bir tanımlayıcı oluşturmasına imkân vermektedir. Türkçe'nin eklemeli bir dil olması program tekrarlanan kelimeleri bulurken aynı köke sahip ancak farklı ek almış kelimeleri farklı olarak kabul etmektedir. Bu nedenle programın *String* işleme kabiliyeti üzerinde geliştirme yapmak mümkündür.

Metadata üreticileri faydalı metadatalar üretme potansiyeline sahiptirler. Ancak otomatik üreticiler tarafından oluşturulmuş metadatalar son ürün olarak düşünülmemelidir. Daha önce de belirtildiği gibi metadata üretme işinde sadece bilgisayar faktörü yeterli kalmamakta insan zekâsının da müdahale etmesi gereken durumlar bulunmaktadır. Bu sebeple metadata üretme işinde üreticiler ve insanların ortak çalışması gerekmektedir. Üretme işleminde yöntem olarak önceki bölümlerde belirtilen Toplama (*Harvesting*) ve Özütleme (*Extraction*) metotlarının beraber kullanımı daha kaliteli metadata çıkarılmasına yardımcı olacaktır.
Metadata üreticileri ile ilgili yukarıda anlatılan olumlu yönlerin yanında geliştirilmesi gereken bazı eksik yönler de bulunmaktadır. Buna göre:

- Metadata üreticilerinin, sadece metadata takılarının bulunduğu en üst seviye sayfada (ana sayfa) etkin biçimde çalışabilme yetenekleri geliştirilmelidir.
- Metinsel yönden fakir sayfalarda özellikle Özütleme (*Extraction*) metodu ile metadata üretmek oldukça zordur. Sayfalarda bulunan resimleri yorumlayabilecek akıllı üreticiler ile ilgili çalışmalar metadata üretme etkinlik kazandırabilir.

Metadata üreticilerinin geliştirilmesi yanında metadata'nın doğru ve belirli bir standarda uygun kullanımını zorunlu kılan düzenleyici bir kuruluşun oluşturulması, Web üzerinde arama motorlarının karşılaştığı uygunsuz metadata kullanımı probleminin çözümünü kolaylaştıracaktır.

Halen piyasada yaygın olarak kullanılan birçok farklı metadata standardı bulunmaktadır. Çok fazla metadata standardının olması kavram karmaşasına neden olmaktadır. Metadata ile ilgili son kullanıcıya da hitap edebilecek basit ve genel bir standardın benimsenmesi gerekmektedir.

Metadata üreticileri belirli algoritmalara göre çalıştıkları için insanlar kadar yorumlama yeteneğine sahip değildirler. O bakımdan metadata üretiminde insan müdahalesi her zaman gerekmektedir. Gelecek yıllarda geliştirilecek kendi kendine öğrenebilen yapay zekâ sistemleri ile insan müdahalesine gerek olmaksızın çalışabilecek programlar yapmanın mümkün olabileceği değerlendirilmektedir.

KAYNAKLAR LİSTESİ

[1] AL U., KÜÇÜK M.E., Üst Veri Standartları ve Uygulamaları, Hacettepe Üniversitesi Edebiyat Fakültesi Dergisi 2003 / Cilt: 20 / Say›: 1 / ss.167–185

[2] BODOFF D. and PATRİCK C.K. Hung, MORDECHAİ Ben-Menachem, Web Metadata Standarts, IEEE Software 2005

[3] CAPELLE C. ,NEJDL W., WOLPERS M., The RDF Schema Specification Revisited, Institut für Technische Informatik Rechnergestützte Wissensverarbeitung Universität Hannover 2000

[4] CANDAN S., LIU H., SUVARNA R., Resource Description Framework: Metadata and Its Applications, Department of Computer Science & Engineering Arizona State University

[5] ÇAMOĞLU K., String İşlemlerinde Performansı Artırmak, http://www.csharpnedir.com/makalegoster.asp?MId=153 Son Ziyaret Tarihi: 05.07.2007

[6] DACONTA Michael C. , OBRST Leo J. Ve SMITH Kevin T. , The Semantic Web, 281s. , 2003

[7] DAY M., Collecting and Preserving the World Wide Web, University of Bath

[8] DEMİRKOL Z. ASP.NET ISBN: 975–6477–02–4 İkinci Basım 580s

[9] DOERR, M., 2001, Semantic Problems of Thesaurus Mapping, Journal of Digital Information, Volume 1 Issue 8 Article No. 52, 2001-03-26

[10] GREENBERG J. ,SPURGİN K., CRYSTAL A. , Functionalities for automatic metadata generation applications: a survey of metadata experts' opinions University of North Carolina

[11] HODGE G., Metadata made simpler, Bethesda, MD: NISO, 2001, s. 5-6

[12] NEİL, S., “Metadata: the matchmaker of the web”, PC Week, 05/04/1998, Vol.15, Issue 18, s.87.

[13] NOUFAL P.P, Metadata: Automatıc generatıon and extractıon , Indian Institute of Technology Kanpur

[14] RAMACHANDRAN S., DOCTOR G. , Metadata Overview: Standards and Interoperability,Paper Submitted to INTERNATIONAL CONFERENCE ON Semantic Web and Digital Libraries (ICSD 2007)

[15] SİMEONİ F. , The Case for Metadata Harvesting, University of STRATHCLYDE
http://hairst.cdlr.strath.ac.uk/documents/MetadataHarvesting.pdf

[16] SNİJDER R., Metadata Standards and Information Analysis: A Survey of Current Metadata Standards and the Underlying Models , 2001

[17] TONTA Y. , Bilgi Erişim Sorunu, 9–20 Nisan tarihlerinde Hatay'da düzenlenen "21. Yüzyıla Girerken Enformasyon Olgusu" konulu seminer bildirisi.

[18] VARIAN R. H. , Universal Access to Information,
Communıcatıons of The ACM October 2005/Vol. 48, No. 10

[19] WEİBEL S. , MİLLER E. , An Introduction to Dublin Core
http://www.xml.com/pub/a/2000/10/25/dublincore/index.html

[20] YONAİTİS, Robert B., Understanding accessibility: a guide to achieving compliance on web sites and intranets, New Hamphire:HiSoftware Publishing, 2002, s.100

[21] Metadata brings knowledge management full circle: a white paper", Concord, NH: Hiawatha Island Software Company, 2001 s.3.

[22] Regular Expressions Cheat Sheet
http://www.ilovejackdaniels.com/cheat-sheets/regular-expressions-cheat-sheet/
Son ziyaret Tarihi: 05.08.2007

[23] Dublin Core Elements
http://library.princeton.edu/departments/tsd/katmandu/html/dcrelation.html
Son Ziyaret Tarihi: 03.06.2007

[24] Ubiquitous Web Applications
http://www.w3.org/2007/uwa/
Son Ziyaret Tarihi: 15.06.2007

[25] XML Nedir?
http://www.acemiyim.net/dokuman/xml/xml-nedir.htm
Son Ziyaret Tarihi: 10.05.2007

[26] Resource Description Framework
http://www.cs.rpi.edu/~puninj/XMLJ/classes/class8/Overview.html
Son Ziyaret Tarihi: 20.07.2007

[27] XML Tutorial
http://www.javacommerce.com/displaypage.jsp?name=intro.sql&id=18238
Son Ziyaret Tarihi: 03.05.2007

[28] Türkiye'de İnternet Kullanıcısı İstatistikleri
http://www.internetworldstats.com/eu/tr.htm
Son Ziyaret Tarihi: 14.08.2007

[29] What search-engines support the Dublin Core Metadata Element Set?
http://www.dublincore.org/resources/faq/#whatsearchenginessupport

Son Ziyaret Tarihi: 14.08.2007

[30] Encoded Archival Description (EAD)
http://www.archivists.org/saagroups/ead/
Son Ziyaret Tarihi: 25.06.2007

[31] VRA Core 4.0 Outline
http://www.vraweb.org/projects/vracore4/VRA_Core4_Outline.pdf
Son Ziyaret Tarihi: 15.07.2007

[32] ONline Information Exchange Frequently Asked Questions
http://www.bisg.org/onix/onix_faq.html
Son Ziyaret Tarihi: 01.08.2007

[33] Online Geospatial Metadata Resources
http://www.fgdc.gov/metadata/online-metadata-resources
Son Ziyaret Tarihi: 16.08.2007

[34] What is Meta?
http://en.wikipedia.org/wiki/Meta
Son Ziyaret Tarihi: 19.08.2007

EKLER

Default.aspx.cs Sınıfı

Bu kısım, aspx sayfasının arkasında çalışan kod bölümüdür. Metin kutusuna URL girildikten sonra yapılan işlemler burada belirtilen koda göre olmaktadır.

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Net;
using System.Text;
using Searchs;
using System.Drawing;
using System.ComponentModel;
using System.Web.SessionState;
using System.Xml;
using System.IO;
using System.Text.RegularExpressions;
using System.Threading;

public partial class _Default : System.Web.UI.Page
{
    public string title, yazar, konu, kapsam, contributor, iliski, dil, tanimlayici, tanim,
yayinci, kaynak, tip, tarih, bicim, copyright;


    public static bool turkceSite = false;

    public static bool altButton = false;

    public static bool ustButton = false;

    protected void Page_Load(object sender, EventArgs e)
    {

        this.lblKonu.Attributes.Add("onclick",
"window.open('sonuc.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblContributor.Attributes.Add("onclick",
"window.open('contributor.htm',null,'height=450, width=450,status= no, resizable=
no, scrollbars=no, toolbar=no,location=no,menubar=no ');");
```

```
        this.lblKapsam.Attributes.Add("onclick",
"window.open('kapsam.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblYazar.Attributes.Add("onclick",
"window.open('yazar.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblTarih.Attributes.Add("onclick",
"window.open('tarih.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblTanim.Attributes.Add("onclick",
"window.open('tanim.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblBicim.Attributes.Add("onclick",
"window.open('bicim.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblTanimlayici.Attributes.Add("onclick",
"window.open('tanimlayici.htm',null,'height=450, width=450,status= no, resizable=
no, scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblDil.Attributes.Add("onclick", "window.open('dil.htm',null,'height=450,
width=450,status= no, resizable= no, scrollbars=no,
toolbar=no,location=no,menubar=no ');");
        this.lblYayinci.Attributes.Add("onclick",
"window.open('yayinci.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblRelation.Attributes.Add("onclick",
"window.open('iliski.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblCopyright.Attributes.Add("onclick",
"window.open('rights.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblKaynak.Attributes.Add("onclick",
"window.open('kaynak.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblKonu.Attributes.Add("onclick",
"window.open('konu.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblTitle.Attributes.Add("onclick",
"window.open('baslik.htm',null,'height=450, width=450,status= no, resizable= no,
scrollbars=no, toolbar=no,location=no,menubar=no ');");
        this.lblTip.Attributes.Add("onclick", "window.open('tip.htm',null,'height=450,
width=450,status= no, resizable= no, scrollbars=no,
toolbar=no,location=no,menubar=no ');");

    }
    protected void ButtonParse_Click(object sender, EventArgs e)
    {

        ustButton = true;

        StringBuilder str = new StringBuilder();
```

```
        LabelMetaResult.Text = "";
        LabelError.Text = "";

        str.Append("<table cellpadding=\"0\" cellspacing=\"0\" border=\"1px\"
width=\"100%\"><tr><td>Adı</td><td>İçerik</td></tr>");
        try
        {

            string URL = TextBoxUrl.Text;

            if (HtmlMetaParser.urlKontrol(URL).ToLower() == "tr") turkceSite = true;

            WebClient client = new WebClient();
            UTF8Encoding enc = new UTF8Encoding();
            string fileContents;

            if (turkceSite == true)

                fileContents=HtmlMetaParser.GetHttpPage(URL);

            else


             fileContents = enc.GetString(client.DownloadData(URL));


            byte[] b = client.DownloadData(URL);
            string strCon = System.Text.Encoding.Unicode.GetString(b);
            string strCon2 = HtmlMetaParser.ByteToString(b);
            string denemestringi = WebFetch.yaziDondur(URL);
            HtmlMetaParser.Parse(fileContents).ForEach(delegate(HtmlMeta n)
            {
             str.Append(String.Format("<tr><td>{0}</td><td>{1}</td></tr>", n.Name ==
"" ? "*" : n.Name.ToUpper(), n.Content));
            });

            str.Append("</table><br/><br/>");

            HtmlMetaParser.Parse(fileContents).ForEach(delegate (HtmlMeta k){

                if (k.Name.ToLower() == "language") this.txtDil.Text =
k.Content.ToString();
                if (k.Name.ToLower() == "copyright") this.txtCopyright.Text =
k.Content.ToString();
                if (k.Name.ToLower() == "author") this.txtYazar.Text =
k.Content.ToString();
                if (k.Name.ToLower() == "keywords") this.txtKapsam.Text =
k.Content.ToString();
                if (k.Name.ToLower()=="description")
this.txtTanim.Text=k.Content.ToString();
```

```
            if (k.Name.ToLower() == "contributor") this.txtContributor.Text =
k.Content.ToString();
            if (k.Name == "") this.txtBicim.Text = k.Content.ToString();
            if (k.Name == "") this.txtBicim.Text = k.Content.ToString();
        });

        Response.Write(str.ToString());

        if (cb_Linkler.Checked) {  LinkParser.ParseLinks(fileContents); }

        string[] kelimeTutucu =
CleanHtml.strinbBol(CleanHtml.Clean(fileContents)).ToArray();

        Response.Write(kelimeTutucu.Length.ToString()+ " kelime bulundu" +
"</br>");

        string[] kelimeSpektrum = new string[kelimeTutucu.Length];
        int[] kelimeSayici=new int[kelimeTutucu.Length];

        TekrarKelime.KelimeBulucu(kelimeTutucu);

        HtmlMetaParser.boldBul(fileContents);
        this.txtTitle.Text = HtmlMetaParser.titleBul(fileContents);
        this.txtTanimlayici.Text = this.TextBoxUrl.Text;

      }
      catch (Exception exp)
      {
        LabelError.Text = exp.Message;
        Response.Write(exp);
      }
    }
    protected void btnOlustur_Click(object sender, EventArgs e)
    {


      altButton = true;

      if (rb_RDF.Checked == true) {

        StringBuilder rdfOlustur = new StringBuilder();
        rdfOlustur.Append(String.Format("<!DOCTYPE rdf:RDF SYSTEM
\"http://dublincore.org/2000/12/01-dcmes-xml-dtd.dtd\">\n"));

        rdfOlustur.Append(String.Format("<rdf:RDF
xmlns:rdf=\"http://w3.org/1999/02/22-rdf-syntaz-ns# xmlns:dc=\"
http://purl.org/dc/elements/1.1/\">\n"));

        if (txtTanimlayici.Text == "") goto rbir;
```

```
            rdfOlustur.Append(String.Format("<rdf:Description rdf:about=\"{0}\" >\n",
this.txtTanimlayici.Text));

        rbir:
            if (txtTitle.Text == "") goto riki;
            rdfOlustur.Append(String.Format("<dc:title>{0} </dc:title>\n",
this.txtTitle.Text));
        riki:
            if (txtYazar.Text == "") goto ruc;
            rdfOlustur.Append(String.Format("<dc:author>{0}</dc:author>\n",
this.txtYazar.Text));
        ruc:
            if (txtKonu.Text == "") goto rdort;
            rdfOlustur.Append(String.Format("<dc:subject>{0}</dc:subject>\n",
this.txtKonu.Text));
        rdort:
            if (txtTanim.Text == "") goto rbes;
            rdfOlustur.Append(String.Format("<dc:description>{0}</dc:description>\n",
this.txtTanim.Text));
        rbes:
            if (txtYayinci.Text == "") goto ralti;
            rdfOlustur.Append(String.Format("<dc:publisher>{0}</dc:publisher>\n",
this.txtYayinci.Text));
        ralti:
            if (txtContributor.Text == "") goto ryedi;
            rdfOlustur.Append(String.Format("<dc:contributor>{0}</dc:contributor>\n",
this.txtContributor.Text));
        ryedi:
            if (txtTarih.Text == "") goto rsekiz;
            rdfOlustur.Append(String.Format("<dc:date>{0}</dc:date>\n",
this.txtTarih.Text));
        rsekiz:
            if (txtTip.Text == "") goto rdokuz;
            rdfOlustur.Append(String.Format("<dc:type>{0}</dc:type>\n",
this.txtTip.Text));
        rdokuz:
            if (txtBicim.Text == "") goto ron;
            rdfOlustur.Append(String.Format("<dc:format>{0}</dc:format>\n",
this.txtBicim.Text));
        ron:
            if (txtTanimlayici.Text == "") goto ronbir;
            rdfOlustur.Append(String.Format("<dc:identifier>{0}</dc:identifier>\n",
this.txtTanimlayici.Text));
        ronbir:
            if (txtKaynak.Text == "") goto roniki;
            rdfOlustur.Append(String.Format("<dc:source>{0}</dc:source>\n",
this.txtKaynak.Text));
        roniki:
            if (txtDil.Text == "") goto ronuc;
```

```
            rdfOlustur.Append(String.Format("<dc:language>{0}</dc:language>\n",
this.txtDil.Text));
        ronuc:
            if (txtIliski.Text == "") goto rondort;
            rdfOlustur.Append(String.Format("<dc:relation>{0}</dc:relation>\n",
this.txtIliski.Text));
        rondort:
            if (txtKapsam.Text == "") goto ronbes;
            rdfOlustur.Append(String.Format("<dc:coverage>{0}</dc:coverage>\n",
this.txtKapsam.Text));
        ronbes:
            if (txtCopyright.Text == "") goto ronalti;
            rdfOlustur.Append(String.Format("<dc:rights>{0}</dc:rights>\n",
this.txtCopyright.Text));
        ronalti:
            rdfOlustur.Append(String.Format("</rdf:Description>\n"));
            rdfOlustur.Append(String.Format("</rdf:RDF>\n"));

            tb_Sonuc.Text = rdfOlustur.ToString();

        }

        if (rb_DColustur.Checked==true)
        {

            if (turkceSite == false)
            {

                XmlTextWriter XmlYaz = new
XmlTextWriter(Server.MapPath("dublinCore.xml"), Encoding.UTF8);

                XmlYaz.WriteStartDocument();
                XmlYaz.Formatting = Formatting.Indented;
                XmlYaz.Indentation = 4;


                XmlYaz.WriteStartElement("record");
                XmlYaz.WriteAttributeString("xmlns:dc",
"http://purl.org/dc/elements/1.1");
                if (txtTitle.Text == "") goto zero;
                XmlYaz.WriteElementString("dc:title", this.txtTitle.Text);

            zero:
                if (txtYazar.Text == "") goto Bir;
                XmlYaz.WriteElementString("dc:creator", this.txtYazar.Text);
            Bir:
                if (txtKonu.Text == "") goto Iki;
                XmlYaz.WriteElementString("dc:subject", this.txtKonu.Text);
            Iki:
                if (txtTanim.Text == "") goto Uc;
```

```
            XmlYaz.WriteElementString("dc:description", this.txtTanim.Text);
        Uc:
            if (txtYayinci.Text == "") goto Dort;
            XmlYaz.WriteElementString("dc:publisher", this.txtYayinci.Text);
        Dort:
            if (txtContributor.Text == "") goto Bes;
            XmlYaz.WriteElementString("dc:contributor", this.txtContributor.Text);
        Bes:
            if (txtTarih.Text == "") goto Alti;
            XmlYaz.WriteElementString("dc:date", this.txtTarih.Text);
        Alti:
            if (txtTip.Text == "") goto Yedi;
            XmlYaz.WriteElementString("dc:type", this.txtTip.Text);
        Yedi:
            if (txtBicim.Text == "") goto Sekiz;
            XmlYaz.WriteElementString("dc:format", this.txtBicim.Text);
        Sekiz:
            if (txtTanimlayici.Text == "") goto Dokuz;
            XmlYaz.WriteElementString("dc:identifier", this.txtTanimlayici.Text);
        Dokuz:
            if (txtKaynak.Text == "") goto On;
            XmlYaz.WriteElementString("dc:source", this.txtKaynak.Text);
        On:
            if (txtDil.Text == "") goto Onbir;
            XmlYaz.WriteElementString("dc:language", this.txtDil.Text);
        Onbir:
            if (txtIliski.Text == "") goto Oniki;
            XmlYaz.WriteElementString("dc:relation", this.txtIliski.Text);
        Oniki:
            if (txtKapsam.Text == "") goto Onuc;
            XmlYaz.WriteElementString("dc:coverage", this.txtKapsam.Text);
        Onuc:
            if (txtCopyright.Text == "") goto Ondort;
            XmlYaz.WriteElementString("dc:rights", this.txtCopyright.Text);
        Ondort:
            XmlYaz.WriteEndElement();
            XmlYaz.WriteEndDocument();
            XmlYaz.Close();
        }

        else
        {

            XmlTextWriter XmlYaz = new
XmlTextWriter(Server.MapPath("dublinCore.xml"),
Encoding.GetEncoding("windows-1254"));

            XmlYaz.WriteStartDocument();
            XmlYaz.Formatting = Formatting.Indented;
            XmlYaz.Indentation = 4;
```

```
            XmlYaz.WriteStartElement("record");
            XmlYaz.WriteAttributeString("xmlns:dc",
"http://purl.org/dc/elements/1.1");
            if (txtTitle.Text == "") goto Tzero;
            XmlYaz.WriteElementString("dc:title", this.txtTitle.Text);

        Tzero:
            if (txtYazar.Text == "") goto TBir;
            XmlYaz.WriteElementString("dc:creator", this.txtYazar.Text);
        TBir:
            if (txtKonu.Text == "") goto TIki;
            XmlYaz.WriteElementString("dc:subject", this.txtKonu.Text);
        TIki:
            if (txtTanim.Text == "") goto TUc;
            XmlYaz.WriteElementString("dc:description", this.txtTanim.Text);
        TUc:
            if (txtYayinci.Text == "") goto TDort;
            XmlYaz.WriteElementString("dc:publisher", this.txtYayinci.Text);
        TDort:
            if (txtContributor.Text == "") goto TBes;
            XmlYaz.WriteElementString("dc:contributor", this.txtContributor.Text);
        TBes:
            if (txtTarih.Text == "") goto TAlti;
            XmlYaz.WriteElementString("dc:date", this.txtTarih.Text);
        TAlti:
            if (txtTip.Text == "") goto TYedi;
            XmlYaz.WriteElementString("dc:type", this.txtTip.Text);
        TYedi:
            if (txtBicim.Text == "") goto TSekiz;
            XmlYaz.WriteElementString("dc:format", this.txtBicim.Text);
        TSekiz:
            if (txtTanimlayici.Text == "") goto TDokuz;
            XmlYaz.WriteElementString("dc:identifier", this.txtTanimlayici.Text);
        TDokuz:
            if (txtKaynak.Text == "") goto TOn;
            XmlYaz.WriteElementString("dc:source", this.txtKaynak.Text);
        TOn:
            if (txtDil.Text == "") goto TOnbir;
            XmlYaz.WriteElementString("dc:language", this.txtDil.Text);
        TOnbir:
            if (txtIliski.Text == "") goto TOniki;
            XmlYaz.WriteElementString("dc:relation", this.txtIliski.Text);
        TOniki:
            if (txtKapsam.Text == "") goto TOnuc;
            XmlYaz.WriteElementString("dc:coverage", this.txtKapsam.Text);
        TOnuc:
            if (txtCopyright.Text == "") goto TOndort;
            XmlYaz.WriteElementString("dc:rights", this.txtCopyright.Text);
        TOndort:
```

```
            XmlYaz.WriteEndElement();
            XmlYaz.WriteEndDocument();
            XmlYaz.Close();

        }

        string strScript = "";
        if (!this.IsStartupScriptRegistered("Baslangic"))
        {
            strScript = "<script language='JavaScript'>";

            strScript += "window.open('dublinCore.xml ";
            strScript += "','anycontent',";
            strScript += "'width=800"; // Genişlik
            strScript += ",height=600"; // Yükseklik
            strScript += ",scrollbars=1"; // Scrollbar
            strScript += "');";
            strScript += "</script>";
            this.RegisterStartupScript("Baslangic", strScript);

        }
    }

    if (rb_takiOlustur.Checked==true) {

        StringBuilder takiOlustur = new StringBuilder();
        if (txtTitle.Text == "") goto Rzero;
        takiOlustur.Append(String.Format("<meta name=\"DC.Title\"
content=\"{0}\">\n", this.txtTitle.Text));
    Rzero:
        if (txtYazar.Text == "") goto Rbir;
        takiOlustur.Append(String.Format("<meta name=\"DC.Creator\"
content=\"{0}\">\n", this.txtYazar.Text));
    Rbir:
        if (txtKonu.Text == "") goto Riki;
        takiOlustur.Append(String.Format("<meta name=\"DC.Subject\"
content=\"{0}\">\n", this.txtKonu.Text));
    Riki:
        if (txtTanim.Text == "") goto Ruc;
        takiOlustur.Append(String.Format("<meta name=\"DC.Keywords\"
content=\"{0}\">\n", this.txtTanim.Text));
    Ruc:
        if (txtTanim.Text == "") goto Rdort;
        takiOlustur.Append(String.Format("<meta name=\"DC.Description\"
content=\"{0}\">\n", this.txtTanim.Text));
    Rdort:
        if (txtYayinci.Text == "") goto Rbes;
        takiOlustur.Append(String.Format("<meta name=\"DC.Publisher\"
content=\"{0}\">\n", this.txtYayinci.Text));
    Rbes:
```

```
            if (txtContributor.Text == "") goto Ralti;
            takiOlustur.Append(String.Format("<meta name=\"DC.Contributor\"
content=\"{0}\">\n", this.txtContributor.Text));
        Ralti:
            if (txtTarih.Text == "") goto Ryedi;
            takiOlustur.Append(String.Format("<meta name=\"DC.Date\"
content=\"{0}\">\n", this.txtTarih.Text));
        Ryedi:
            if (txtTip.Text == "") goto Rsekiz;
            takiOlustur.Append(String.Format("<meta name=\"DC.Type\"
content=\"{0}\">\n", this.txtTip.Text));
        Rsekiz:
            if (txtBicim.Text == "") goto Rdokuz;
            takiOlustur.Append(String.Format("<meta name=\"DC.Format\"
content=\"{0}\">\n", this.txtBicim.Text));
        Rdokuz:
            if (txtTanimlayici.Text == "") goto Ron;
            takiOlustur.Append(String.Format("<meta name=\"DC.Identifier\"
content=\"{0}\"\n>", this.txtTanimlayici.Text));
        Ron:
            if (txtKaynak.Text == "") goto Ronbir;
            takiOlustur.Append(String.Format("<meta name=\"DC.Source\"
content=\"{0}\">\n", this.txtKaynak.Text));
        Ronbir:
            if (txtDil.Text == "") goto Roniki;
            takiOlustur.Append(String.Format("<meta name=\"DC.Language\"
content=\"{0}\"\n>", this.txtDil.Text));
        Roniki:
            if (txtIliski.Text == "") goto Ronuc;
            takiOlustur.Append(String.Format("<meta name=\"DC.Relation\"
content=\"{0}\"\n>", this.txtIliski.Text));
        Ronuc:
            if (txtKapsam.Text == "") goto Rondort;
            takiOlustur.Append(String.Format("<meta name=\"DC.Coverage\"
content=\"{0}\"\n>", this.txtKapsam.Text));
        Rondort:
            if (txtCopyright.Text == "") goto Ronbes;
            takiOlustur.Append(String.Format("<meta name=\"DC.Rights\"
content=\"{0}\">\n", this.txtCopyright.Text));
        Ronbes:
            tb_Sonuc.Text = takiOlustur.ToString();

            }
        }
    }
```

Default.aspx Ara yüzü

Sayfanın görsel tasarım kodlarını içermektedir.

```
<%@ Page Language="C#" AutoEventWireup="true"  codePage="1254"
CodeFile="Default.aspx.cs" Inherits="_Default" EnableViewState="true"
UICulture="Auto" Culture="Auto" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
   <title>Metadata İşleyici</title>
</head>
<body bgcolor="gainsboro">
   <form id="form1" runat="server">
   <div>
      <table cellpadding="0" cellspacing="0">
         <tr>
            <td colspan="2">
               Lütfen incelenmesi istenen web adresini giriniz
            </td>
         </tr>
         <tr>
            <td>
               <asp:TextBox ID="TextBoxUrl" runat="server" Width="240px"
CausesValidation="True"></asp:TextBox><br />
               <asp:RegularExpressionValidator runat="server"
ID="RegularExpressionValidatorUrl" ControlToValidate="TextBoxUrl"
                  ValidationExpression="^(ht|f)tp(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-
Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&amp;%\$#_]*)?$"> Geçerli Bir URL Giriniz
</asp:RegularExpressionValidator>
            </td>
            <td valign="top">
               <asp:Button ID="ButtonParse" runat="server" Text="Getir"
OnClick="ButtonParse_Click"></asp:Button>
            </td>
         </tr>
      </table>
   </div>
      <asp:CheckBox ID="cb_Linkler" runat="server" Text="Linkleri Göster"
ToolTip="Sayfada Bulunan Linkleri Gösterir" /><br />
      <asp:Label ID="LabelMetaResult" runat="server"></asp:Label><br />
      <asp:Label ID="LabelError" runat="server" ForeColor="Red"></asp:Label>
      <asp:Panel ID="Panel1" runat="server" BackColor="Transparent"
BorderColor="#400000" Height="327px"
         Width="693px" BorderStyle="None" Wrap="False">
         <asp:Label ID="lblTitle" runat="server" Height="28px" Text="Başlık"
Width="137px" Font-Bold="True" ToolTip="Title"></asp:Label>
         <asp:TextBox ID="txtTitle" runat="server" Width="394px" Height="22px"
></asp:TextBox>
```

```
        <br />
        <asp:Label ID="lblYazar" runat="server" Height="28px" Text="Yazar"
Width="137px" ToolTip="Creator"></asp:Label>
        <asp:TextBox ID="txtYazar" runat="server" Height="22px"
Width="394px"></asp:TextBox>
        <br />
        <asp:Label ID="lblKonu" runat="server" Height="28px" Text="Konu"
Width="137px" ToolTip="Subject "></asp:Label>
        <asp:TextBox ID="txtKonu" runat="server" Height="22px"
Width="394px"></asp:TextBox>
        <br />
        <asp:Label ID="lblTanim" runat="server" Height="28px" Text="Tanım"
Width="137px" ToolTip="Description"></asp:Label>
        <asp:TextBox ID="txtTanim" runat="server" Height="22px" Width="394px"
></asp:TextBox>
        <br />
        <asp:Label ID="lblYayinci" runat="server" Height="28px" Text="Yayıncı"
Width="137px" ToolTip="Publisher"></asp:Label>
        <asp:TextBox ID="txtYayinci" runat="server" Height="22px"
Width="394px"></asp:TextBox>
        <br />
        <asp:Label ID="lblContributor" runat="server" Height="28px" Text="Katkıda
Bulunanlar"
            Width="137px" ToolTip="Contributor"></asp:Label>
        <asp:TextBox ID="txtContributor" runat="server" Height="22px"
Width="394px"></asp:TextBox>
        <br />
        <asp:Label ID="lblTarih" runat="server" Height="28px" Text="Tarih"
Width="137px" ToolTip="Date"></asp:Label>
        <asp:TextBox ID="txtTarih" runat="server" Height="22px" Width="394px"
ToolTip="GG-AA-YYYY GG/AA/YYYY GG.AA.YYYY"></asp:TextBox>
        <asp:RegularExpressionValidator id="RegularExpressionValidator1"
runat="server" ControlToValidate="txtTarih" ValidationExpression="(0[1-9]|[12][0-
9]|3[01])[- /.](0[1-9]|1[012])[- /.](19|20)\d\d"
            ErrorMessage="Tarihi GG/AA/YYYY şeklinde
giriniz!!!"></asp:RegularExpressionValidator>
        <br />
        <asp:Label ID="lblTip" runat="server" Height="28px" Text="Tip"
Width="137px" ToolTip="Type "></asp:Label>
        <asp:TextBox ID="txtTip" runat="server" Height="22px" Width="394px"
ToolTip="Yazı, Resim, Ses, Kolleksiyon vb."></asp:TextBox>
        <br />
        <asp:Label ID="lblBicim" runat="server" Height="28px" Text="Biçim"
Width="137px" ToolTip="Format"></asp:Label>
        <asp:TextBox ID="txtBicim" runat="server" Height="22px" Width="394px"
ToolTip="Dosya Formatı PDF, GIF, EXE vb."></asp:TextBox>
        <br />
        <asp:Label ID="lblTanimlayici" runat="server" Height="28px"
Text="Tanımlayıcı" Width="137px" ToolTip="Identifier"></asp:Label>
```

```
        <asp:TextBox ID="txtTanimlayici" runat="server" Height="22px"
Width="394px" ToolTip="URL, URN, ISBN vb."></asp:TextBox>
        <br />
        <asp:Label ID="lblKaynak" runat="server" Height="28px" Text="Kaynak"
Width="137px" ToolTip="Source"></asp:Label>
        <asp:TextBox ID="txtKaynak" runat="server" Height="22px" Width="394px"
ToolTip="Alıntı ise asıl kaynak" ></asp:TextBox>
        <br />
        <asp:Label ID="lblDil" runat="server" Height="28px" Text="Dil"
Width="137px" ToolTip="Language"></asp:Label>
        <asp:TextBox ID="txtDil" runat="server" Height="22px"
Width="394px"></asp:TextBox>
        <br />
        <asp:Label ID="lblRelation" runat="server" Height="28px" Text="İlişki"
Width="137px" ToolTip="Relation"></asp:Label>
        <asp:TextBox ID="txtIliski" runat="server" Height="22px"
Width="394px"></asp:TextBox>
        <br />
        <asp:Label ID="lblKapsam" runat="server" Height="28px" Text="Kapsam"
Width="137px" ToolTip="Coverage"></asp:Label>
        <asp:TextBox ID="txtKapsam" runat="server" Height="22px"
Width="394px"></asp:TextBox>
        <br />
        <asp:Label ID="lblCopyright" runat="server" Height="28px"
Text="Copyright" Width="137px" ToolTip="Rights"></asp:Label>
        <asp:TextBox ID="txtCopyright" runat="server" Height="22px"
Width="394px" AutoPostBack="True"></asp:TextBox><br />
        <asp:Label ID="Label1" runat="server" Height="28px" Text="Oluşturma
Türü" Width="137px" Font-Bold="True"></asp:Label>
        <asp:RadioButton ID="rb_takiOlustur" runat="server" Checked="True"
GroupName="olusturma"
            Text="HTML Takısı" Font-Bold="True" />
        <asp:RadioButton ID="rb_DColustur" runat="server"
GroupName="olusturma" Text="Dublin Core" Font-Bold="True" />
        <asp:RadioButton ID="rb_RDF" runat="server" Font-Bold="True"
GroupName="olusturma"
            Text="RDF/XML" /><br />
        <br />
        <asp:TextBox ID="tb_Sonuc" runat="server" Height="75px"
TextMode="MultiLine" Width="545px"></asp:TextBox><br />
         <asp:Button ID="btnOlustur" runat="server" Text="Oluştur"
OnClick="btnOlustur_Click"  /><br />
        <br />
        <br />
      </asp:Panel>
           <br />
      <br />
   </form>
</body>
</html>
```

CleanHtml.cs Sınıfı

Sayfanın kaynak kodundaki HTML takılarını temizleyen Clean metodu ile tekrar eden kelimeleri bulan TekrarBul metotlarını içerir.

```
using System;
using System.Text.RegularExpressions;
using System.Diagnostics;
using System.Net;
using System.Web.UI.WebControls;
using System.Collections;
using System.Collections.Generic;


/// <summary>
/// Summary description for CleanHtml.
/// </summary>

namespace Searchs
{
   public class CleanHtml
   {
      public CleanHtml()
      {

      }
      public static string Clean(string Contents)
      {
         if (Contents.IndexOf("startnav") != -1)
         {
            int startnav = Contents.IndexOf("startnav");
            int endnav = Contents.IndexOf("endnav");
            int lengthnav = endnav - startnav;
            string subContents = Contents.Substring(startnav, lengthnav);
            Contents = Contents.Replace(subContents, "");
         }
         Contents = Regex.Replace(Contents,
"<(select|option|script|style|title)(.*?)>((.|\n)*?)</(select|option|script|style|title)>", "
", RegexOptions.IgnoreCase);
         Contents = Regex.Replace(Contents, "&(nbsp|quot|copy|bull);", "");
         Contents = Regex.Replace(Contents, "&amp;", "&");
         Contents = Regex.Replace(Contents, "<[^>]*>", "");
         Contents = Regex.Replace(Contents, "<([\\s\\S])+?>", " ",
RegexOptions.IgnoreCase).Replace("  ", " ");
         Contents = Regex.Replace(Contents, "<[^<>]+>", " ",
RegexOptions.IgnoreCase);
         Contents = Regex.Replace(Contents, "\\W", " ");
         return Contents;
      }
```

```
    public static void TekrarBul(string kaynak)
    {

        Regex rx = new Regex(@"\b(?<word>\w+)\s*(\k<word>)\b",
          RegexOptions.Compiled | RegexOptions.IgnoreCase);


        MatchCollection matches = rx.Matches(kaynak);

        System.Web.HttpContext.Current.Response.Write("</br>");
        System.Web.HttpContext.Current.Response.Write( matches.Count + " Tekrar
eden kelime bulundu" );
        System.Web.HttpContext.Current.Response.Write("</br>");

        foreach (Match match in matches)
        {
            int tekrar=0 ;
            string word = match.Groups["word"].Value;
            tekrar = matches.Count + 1;
            System.Web.HttpContext.Current.Response.Write( word + " kelimesi "+
tekrar + " kere tekrar etti");
            System.Web.HttpContext.Current.Response.Write("</br>");

        }
    }

        public static List<string> strinbBol(string htmlkaynak) {

            List<string> bolunmusKaynak=new List<string>();
            string s = htmlkaynak;
           Regex o = new Regex(@"[\s]+");
            foreach (string ss in o.Split(htmlkaynak))
            {

                bolunmusKaynak.Add(ss);

            }

            return bolunmusKaynak;

        }


    public static object turkcekarakter(string p)
        {
            throw new Exception("Hata");
        }
    }
}
```

HtmlMetaParser.cs Sınıfı

BoldBul, titleBul,urlKontrol ve Parse metotlarını içerir.

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Text.RegularExpressions;
using System.Collections.Generic;
using System.Text;

/// <summary>
/// Summary description for HtmlMetaParser
/// </summary>
public class HtmlMetaParser
{
   public static String ByteToString(byte[] b)
   {
      String t = "";
      for (int i = 0; i < b.Length; i++) t = t + (char)b[i];
      return t;
   }

   public static void boldBul(string kaynakKod) {

      StringBuilder tablocu = new StringBuilder();

      Regex boldBulucu = new Regex(@"(<b>)(?<yazi>(.*))(</b>)",
RegexOptions.IgnoreCase | RegexOptions.ExplicitCapture);

      tablocu.Append("<table cellpadding=\"0\" cellspacing=\"0\" border=\"1px\"
width=\"100%\"><tr><td>BOLD YAZILMIS KELIMELER</td></tr>");
      foreach (Match tut in boldBulucu.Matches(kaynakKod)) {

         tablocu.Append(String.Format("<tr><td>{0}</td></tr>",
tut.Value.ToString()));
               }
      tablocu.Append("</table><br/><br/>");

      System.Web.HttpContext.Current.Response.Write(tablocu);
   }
```

```
    public static string titleBul(string kaynak)
    {

       Regex titleBulucu = new Regex(@"(<title>)(?<yazi>(.*))(</title>)",
       RegexOptions.IgnoreCase | RegexOptions.ExplicitCapture);
        string basi="";
        foreach (Match baslik in titleBulucu.Matches(kaynak))
        {
            basi = baslik.Groups["yazi"].Value.ToString();

        }
        return basi;
        }


    public static string urlKontrol(string url) {

        string sonTaraf = "";
        Regex uzanti = new
Regex(@"(http://www.)(?<adres>(.*))((.))((com)|(edu)|(gov)|(co)|(mil)|(org))((.))(?<t
urk>(.*))(\s)*", RegexOptions.IgnoreCase | RegexOptions.ExplicitCapture);
        foreach (Match kisim in uzanti.Matches(url))
        {
            sonTaraf = kisim.Groups["turk"].ToString();
        }
        return sonTaraf;
    }

    public enum RobotHtmlMeta
    {
        None = 0, NoIndex, NoFollow,
        NoIndexNoFollow
    }
    public static List<HtmlMeta> Parse(string htmldata)
    {
        Regex metaregex =
            new Regex(@"<meta\s*(?:(?:\b(\w|-)+\b\s*(?:=\s*(?:""[^""]*""|'" +
                    @"[^']*'|[^""'<> ]+)\s*)?)*)/?\s*>",
                    RegexOptions.IgnoreCase | RegexOptions.ExplicitCapture);

        List<HtmlMeta> MetaList = new List<HtmlMeta>();

        foreach (Match metamatch in metaregex.Matches(htmldata))
        {
            HtmlMeta mymeta = new HtmlMeta();

            Regex submetaregex =
               new Regex(@"(?<name>\b(\w|-)+\b)\" +
```

```
                @"s*=\s*(""(?<value>" +
                @"[^""]*)""|'(?<value>[^']*)'" +
                @"|(?<value>[^""'<> ]+)\s*)+",
                RegexOptions.IgnoreCase |
                RegexOptions.ExplicitCapture);

        foreach (Match submetamatch in
                submetaregex.Matches(metamatch.Value.ToString()))
        {
            if ("http-equiv" ==
                 submetamatch.Groups["name"].ToString().ToLower())
                mymeta.HttpEquiv =
                 submetamatch.Groups["value"].ToString();

            if (("name" ==
                 submetamatch.Groups["name"].ToString().ToLower())
                 && (mymeta.HttpEquiv == String.Empty))
                mymeta.Name = submetamatch.Groups["value"].ToString();

            if ("scheme" ==
                submetamatch.Groups["name"].ToString().ToLower())
                mymeta.Scheme = submetamatch.Groups["value"].ToString();

            if ("content" ==
                submetamatch.Groups["name"].ToString().ToLower())
            {
                mymeta.Content = submetamatch.Groups["value"].ToString();
                MetaList.Add(mymeta);
            }
        }
    }

    return MetaList;
}

public static RobotHtmlMeta ParseRobotMetaTags(string htmldata)
{
    List<HtmlMeta> MetaList = HtmlMetaParser.Parse(htmldata);

    RobotHtmlMeta result = RobotHtmlMeta.None;
    foreach (HtmlMeta meta in MetaList)
    {
        if (meta.Name.ToLower().IndexOf("robots") != -1 ||
                meta.Name.ToLower().IndexOf("robot") != -1)
        {
            string content = meta.Content.ToLower();
            if (content.IndexOf("noindex") != -1 &&
                content.IndexOf("nofollow") != -1)
            {
                result = RobotHtmlMeta.NoIndexNoFollow;
```

```
                break;
            }
            if (content.IndexOf("noindex") != -1)
            {
                result = RobotHtmlMeta.NoIndex;
                break;
            }
            if (content.IndexOf("nofollow") != -1)
            {
                result = RobotHtmlMeta.NoFollow;
                break;
            }
        }
    }
    return result;
  }
  public static string GetHttpPage(string pageUrl)
  {
    System.Net.WebResponse response = null;

    string strSayfa;
    try
    {
      System.Net.WebRequest request =
System.Net.WebRequest.Create(pageUrl);
        request.Timeout = 10000;
        response = request.GetResponse();
        System.IO.Stream streamReceive = response.GetResponseStream();
        System.Text.Encoding encoding =
System.Text.Encoding.GetEncoding("windows-1254");
        System.IO.StreamReader streamRead = new
System.IO.StreamReader(streamReceive, encoding);
        strSayfa = streamRead.ReadToEnd();

        return strSayfa;
    }
    finally
    {
        if ((response != null))
        {
            response.Close();
        }
    }
  }
}
```

LinkParser.cs Sınıfı

Sayfadaki bağlantıları bulan ParseLinks metodunu içerir.

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Collections;
using System.Text.RegularExpressions;
using System.Text;

/// <summary>
/// Summary description for LinkParser
/// </summary>
public class LinkParser
{
   public static string ParseLinks(string htmlData)
   {
      StringBuilder str = new StringBuilder();
      ArrayList linkLocal = new ArrayList();
      ArrayList linkExternal = new ArrayList();
      string link = String.Empty;

      foreach (Match match in Regex.Matches(htmlData
            ,
@"(?<anchor><\s*(a|area)\s*(?:(?:\b\w+\b\s*(?:=\s*(?:""[^""]*""|'[^']*'|[^""'<>
]+)\s*)?)*)?\s*>)"
            , RegexOptions.IgnoreCase | RegexOptions.ExplicitCapture))
      {

         link = String.Empty;
         foreach (Match submatch in Regex.Matches(match.Value.ToString()
            ,
@"(?<name>\b\w+\b)\s*=\s*(""(?<value>[^""]*)""|'(?<value>[^']*)'|(?<value>[^""'<>
\s]+)\s*)+"
            , RegexOptions.IgnoreCase | RegexOptions.ExplicitCapture))
         {

            if ("href" == submatch.Groups[1].ToString().ToLower())
            {
               link = submatch.Groups[2].ToString();
               break;
            }
         }
```

```
            if (link.IndexOf("#") > -1)
            {
                link = link.Substring(0, link.IndexOf("#"));
            }
            if (link.IndexOf("javascript:") == -1
                && link.IndexOf("mailto:") == -1
                && !link.StartsWith("#")
                && link != String.Empty)
            {
                if ((link.Length > 8) && (link.StartsWith("http://")
                                        || link.StartsWith("https://")
                                        || link.StartsWith("file://")
                                        || link.StartsWith("//")
                                        || link.StartsWith(@"\\")))
                {
                    linkExternal.Add(link);
                }
                else if (link.StartsWith("?"))
                {
                    linkLocal.Add(link);
                }
                else
                {
                    linkLocal.Add(link);
                }
            }
        }

        str.Append("<table cellpadding=\"0\" cellspacing=\"0\" border=\"1px\"
width=\"100%\"><tr><td>Site İçi Linkler</td></tr>");
        foreach (string lnk in linkLocal.ToArray(typeof(string)))
            str.Append(String.Format("<tr><td>{0}</td></tr>", lnk));
        str.Append("</table><br/><br/>");
        str.Append("<table cellpadding=\"0\" cellspacing=\"0\" border=\"1px\"
width=\"100%\"><tr><td>Site Dışı Linkler</td></tr>");
        foreach (string lnk in linkExternal.ToArray(typeof(string)))
            str.Append(String.Format("<tr><td>{0}</td></tr>", lnk));
        str.Append("</table><br/><br/>");
        return str.ToString();
    }
}
```

TekrarKelime.cs Sınıfı

Sayfada tekrar eden kelimeleri bulan KelimeBulucu metodunu içerir.

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Text.RegularExpressions;
using System.IO;
using System.Collections;
using System.Text;

public class TekrarKelime
{


    public static void KelimeBulucu(string[] kelimeDizisi)
    {


        ArrayList diziElemanlari = new ArrayList();


        for (int i = 0; i < kelimeDizisi.Length; i++)
        {
            if (diziElemanlari.Contains(kelimeDizisi[i]) == false)
            {
                diziElemanlari.Add(kelimeDizisi[i]);
            }
        }


        int[] elemanTekrarlari = new int[diziElemanlari.Count];


        for (int i = 0; i < diziElemanlari.Count; i++)
        {
            elemanTekrarlari[i] = 0;
            string siradaki = diziElemanlari[i].ToString();
            for (int j = 0; j < kelimeDizisi.Length; j++)
            {
                if (siradaki == kelimeDizisi[j])
                {
                    elemanTekrarlari[i]++;
```

```
            }
          }
        }

          for (int i = 0; i < diziElemanlari.Count; i++)
        {
            if(elemanTekrarlari[i]>5)
            System.Web.HttpContext.Current.Response.Write("Kelime : " +
diziElemanlari[i].ToString() + " - Tekrar Sayısı : " + elemanTekrarlari[i].ToString() +
"</br>");

        }
    }
}
```