KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

YÜKSEK LİSANS TEZİ

DİJİTAL PDF DOKÜMANLARDAN BİÇİM TANIMA VE FARKLI İÇERİKLERE GİYDİRME: ÖZGEÇMİŞLER ÜZERİNDE DURUM ÇALIŞMASI

ALPER KANTARCI

KOCAELİ 2020

KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

YÜKSEK LİSANS TEZİ

DİJİTAL PDF DOKÜMANLARDAN BİÇİM TANIMA VE FARKLI İÇERİKLERE GİYDİRME: ÖZGEÇMİŞLER ÜZERİNDE DURUM ÇALIŞMASI

ALPER KANTARCI

Doç.Dr. Ahmet SAYAR
Danışman, Kocaeli Üniv. .....

Prof.Dr. Nevcihan DURU
Jüri Üyesi, Kocaeli Üniv. .....

Prof.Dr. İbrahim ÖZÇELİK
Jüri Üyesi, Sakarya Üniv. .....

Tezin Savunulduğu Tarih: 23.01.2020

ÖNSÖZ VE TEŞEKKÜR

Bu tez çalışması, kişilerin özgeçmiş hazırlayarak zamanlarını kaybetmemek ve farklı kabul görmüş formatlarda kişilerin kendi bilgilerine göre kendilerine has özgeçmişler hazırlayabilmesine imkân verecek web tabanlı zeki özgeçmiş tasarımcısı geliştirmek amacıyla gerçekleştirilmiştir.

Lisans ve Yüksek lisans öğrenimim boyunca görüşleri ile çalışmalarıma katkıda bulunan, karşılaştığım her zorlukta desteğini ve zamanını esirgemeyen hocam Doç.Dr. Ahmet SAYAR'a sonsuz teşekkürlerimi sunarım.

Tez çalışmamın tüm aşamalarında bilgi ve destekleriyle katkıda bulunan hocam Dr.Öğr.Üyesi Süleyman EKEN'e teşekkür ediyorum.

Hayatım boyunca bana güç veren en büyük destekçilerim, her aşamada sıkıntılarımı ve mutluluklarımı paylaşan sevgili babam Türhan KANTARCI, annem Elmas KANTARCI ve kardeşlerim Hüsein KANTARCI ile Onur KANTARCI'ya teşekkürlerimi sunarım.

Tez çalışması, Kocaeli Üniversitesi Bilimsel Araştırma Projeleri Koordinasyonu Birimi (BAP) tarafından 2018/136 nolu proje kapsamında desteklenmiştir. Desteklerinden dolayı teşekkür ederim.

Ocak – 2020 Alper KANTARCI

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

Kısaltmalar

API	:	Application Programming Interface (Yazılım Programlama Arayüzü)
ASCII	:	American Standard Code for Information Interchange (Bilgi Değişimi İçin Amerikan Standart Kodu)
CV	:	Curriculum Vitae (Özgeçmiş)
HTML	:	Hypertext Markup Language (Hiper Metin İşaret Dili)
ISO	:	International Standards Organization (Uluslararası Standartlaştırma Organizasyonu)
LZW	:	Lempel–Ziv–Welch Universal Lossless Data Compression Algorithm (Evrensel Kayıpsız Veri Sıkıştırma Algoritması)
NLP	:	Natural Language Processing (Doğal Dil İşleme)
OCR	:	Optical Character Recognition (Optik Karakter Tanıma)
PDF	:	Portable Document Format (Taşınabilir Belge Formatı)
XML	:	Extensible Markup Language (Genişletilebilir İşaretleme Dili)
XSD	:	XML Schema Definition (XML Şeması Tanımı)

DİJİTAL PDF DOKÜMANLARDAN BİÇİM TANIMA VE FARKLI İÇERİKLERE GİYDİRME: ÖZGEÇMİŞLER ÜZERİNDE DURUM ÇALIŞMASI

ÖZET

Çoğu bilgisayar işleminin merkezinde yer alan toplu kategorizasyona ilişkin olarak bilgi geri çağırmayı etkileyen iki tür ilgili veri vardır: yapısal veriler ve yapılandırılmamış veriler. Yapılandırılmış veriler, ilişkisel bir veritabanına dahil edilmesi gibi yüksek derecede organizasyona sahip bilgileri ifade eder. Bununla birlikte, yapılandırılmamış veriler kendi iç yapısına sahip olabilir, ancak bir e-tabloya veya veritabanına tam olarak karşılık gelmezler. Özgeçmişler bu tür verilerdir. Genelde PDF (Portable Document Format, Taşınabilir Belge Formatı) formatında sunulan özgeçmişler, PDF etiketleme özelliği kullanılarak yapısal hale getirilebilir; fakat çoğu PDF verisi etiketlenmemiş ve yapısal olmayan haldedir. Teknik olmayan iş dünyası kullanıcıları ve veri analistlerinin bu tür kapalı kutularla başa çıkmaları çok zordur.

Bu çalışma kapsamında, kişilerin özgeçmiş hazırlayarak zamanlarını kaybetmemek ve farklı kabul görmüş formatlarda kişilerin kendi bilgilerine göre kendilerine has özgeçmişler hazırlayabilmesine imkân verecek web tabanlı zeki özgeçmiş tasarımcısı geliştirildi. PDF dokümanlarının içerik yapısı, metin verisi ve bu verinin yazı tipi ve dokümandaki lokasyon bilgileri çıkartıldı ve elde edilen bu bilgiler okuma sırasına göre belirli yapılara dönüştürülerek önceden tanımlanmış olan XML (Extensible Markup Language, Genişletilebilir İşaretleme Dili) tabanlı özgeçmiş tasarımı oluşturuldu. Elde edilen bu tasarımlar kullanılarak kişisel PDF dökümanları oluşturuldu. PDF analizi ve PDF oluşturma işlemi, Java iText-pdf kütüphanesi yardımıyla gerçekleştirildi. Tasarım verileri arayüz aracılığıyla kullanıcıya sunularak kullanıcı istediği tasarımı kendi dökümanını oluştururken seç ve uygula yaklaşımıyla aktarabilmektedir.

PDF dokümanından elde edilen şablonun XML formatında kaydedilmesi ve farklı içeriklere uyarlama aşamasında, kaydedilmiş hazır XML formatındaki şablonların kullanılması öngörüldü. XML formatındaki şablonların otomatik oluşturulabilmesi ve sonradan doğruluğunun test edilebilmesi için XSD (XML Schema Definition, XML Şeması Tanımı) tanımlandı. Geliştirilen uygulama ile özgeçmişlerin otomatik biçimlerinin tanınması ve farklı içeriklerin adaptasyonu sağlandı.

Anahtar Kelimeler: Bilgi Çıkarımı, Doküman Analizi ve Tanıma, PDF, XML, XSD.

STRUCTURE RECOGNITION ON DIGITAL PDF DOCUMENTS AND ADAPTING TO DIFFERENT CONTENTS: CASE STUDY ON RESUMES

ABSTRACT

With respect to the mass categorization that is central to most computer operations, there are two types of relevant data which affect speed of assimilation as well as information recall: structured data and unstructured data. Structured data refers to information with a high degree of organization, such that inclusion in a relational database. However, unstructured data may have its own internal structure, but does not conform neatly into a spreadsheet or database. CVs (Curriculum Vitae, Özgeçmiş) are this kind of data. Typically, CVs presented in PDF format can be structured using the PDF tagging feature, however most PDF data is untagged and unstructured. It is very difficult for non-technical business users and data analysts to deal with such closed boxes.

Within the scope of this study, a web based smart resume designer was developed which will allow people gain time while creating their own resumes according to their own information in different accepted formats. The content structure of the PDF documents, the text data and the font and location information of this data were extracted and the information obtained was converted into certain structures in the order of reading and a predefined XML based resume template was created. Personal PDF documents are created using this template. PDF analysis and PDF creation was done directly by accessing the content stream of the PDF document with the help of the iText-pdf library, which is the Java library. Presentation templates is served to end-user on a desktop application with a GUI and users can select any metadata to create own document with select-and-apply approach.

It is predicted that the template obtained from the PDF document will be saved in XML format and the templates in the ready-made XML format will be used for adaptation to different contents. The XML schema (XSD-xml schema definition) is defined for the automatic creation of templates in XML format and subsequent testing of their accuracy. With the application developed, automatic forms of resumes were recognized and different contents were adapted.

Keywords: Information Extraction, Document Analysis and Recognition, PDF, XML, XSD.

GİRİŞ

İş bulmak için atılacak ilk adım profesyonelce yazılmış bir özgeçmiştir. Özgeçmişinin içeriğini, isim, soy isim (kişisel bilgiler), iletişim bilgileri, eğitim durumu, iş tecrübesi deneyimler, referanslar, özel zevkler gibi konuların özetleri oluşturur. Özgeçmiş doğru, anlaşılır, açık ve kısa olmalıdır ki dikkat çekilebilsin. Etkili bir özgeçmiş hazırlamak için uzmanlar bazı noktalara dikkat edilmesi gerektiğini vurgulamaktalar [1]. Bunlardan bazıları:

- Özgeçmiş en fazla 2 sayfa olmalıdır.
- Uzun paragraflardan kaçınılmalıdır.
- Genellikle "Times New Roman" veya "Arial" gibi kolay okunabilen karakterler kullanılmalı, 11 ya da 12 punto ile yazılmalıdır.
- Kelime ve cümlelerin altı çizilmemelidir.
- Ters kronolojik sıra takip edilmemelidir.
- Gereksiz bilgiler eklemekten kaçınılmalıdır.
- Takım elbiseli çekilmiş fotoğraflar eklenmelidir.
- Silik, bulanık, arka planı karışık fotoğraflar eklenmemelidir.

Genel olarak bakıldığında özgeçmiş hazırlamak ince işçilik isteyen bir süreçtir. Bu yüzden hemen özgeçmiş hazırlayıp başvurularda bulunmak yerine zamana yayıp genel geçer kurallara uygun olarak ilgi çekici bir özgeçmiş hazırlamak yapılan başvurulara geri dönüşler almayı epey kolaylaştırmaktadır. Yukarıda listelenen veya uzmanlar tarafından söylenen kriterlerde özgeçmişler hazırlamak özellikle Office programlarını kullanmayı tam bilmeyen kimseler tarafından hem çok sıkıcı hem de çok zaman alan bir süreç olmaktadır. Biz de bu çalışma kapsamında, kişilerin özgeçmiş hazırlayarak zamanlarını kaybetmemek ve farklı kabul görmüş formatlarda kişilerin kendi bilgilerine göre kendilerine has özgeçmişler hazırlayabilmesine imkân verecek web tabanlı zeki özgeçmiş tasarımcısı geliştirdik. Yaptığımız araştırmalara göre literatürde PDF özgeçmiş dokümanlarını kullanarak kullanıcılara özgü yeni özgeçmişlerin hazırlanmasını sağlayan bir çalışmaya rastlamadık.

İşletim sisteminden bağımsız olması nedeniyle, PDF formatı elektronik belgelerin paylaşılması, arşivlenmesi, alınması ve yazdırılması için yaygın olarak kullanılır [2, 3]. Yaygın olarak kullanıldığı alanlar ve faydaları yanısıra, içerik ve yapı analizi açısından dezavantajlara da sahiptir [2]. Sonuç olarak PDF formatlı dokümanlarda sunulan bilgiler, otomatik olarak okunması ve yeniden kullanılması gibi işlem gerektiren, özgeçmiş gibi karar alma uygulamaları için elverişli değildir [2]. Günümüzde yaygın olarak kullanılmasına rağmen PDF belge içeriği iç yapısı hakkında da bilgi eksikliği var [4, 5]. PDF belge yapısını otomatik olarak analiz etme ve tanıma, ilgili bilgilerin çıkarılması ve bu bilgilerin hem yapı hem de anlamsal biçimlerde ayrıştırılması, paylaşım, bilgi arama, karar verme ve diğer çeşitli amaçlar için büyük önem taşımaktadır [2]. Çoğu PDF belgesi etiketlenmemişti ve temel üst düzey doküman mantıksal yapısal bilgisine sahip olmadığı için, belgelerin yeniden kullanılması veya değiştirilmesini zorlaştırmaktadır [6]. PDF formatlı özgeçmiş belgelerinin yapı analizi en basit yapıya sahip olanları için bile zorlu bir işlemdir. PDF belgelerini, resim veya HTML'e dönüştürüp sonra OCR (Optical Character Recognition, Optik Karakter Tanıma) vs. gibi tekniklerle işlemek yerine, içeriği doğrudan PDF`ten çıkarmak ve analiz etmek daha kolay ve kesindir [4]. Bu çalışmada PDF formatındaki CV dokümanlarının tasarım şablonlarının otomatik olarak tanınması, bu şablonların daha sonra kullanılabilirliğini sağlamak amaçlı XML formatına dönüştürülüp kaydedilmesi ve farklı içeriklere uyarlanması gerçekleştirildi. XML formatındaki şablonların otomatik oluşturulabilmesi ve sonradan doğruluğunun test edilebilmesi için önceden XML şeması tasarlandı.

Literatürde direk önerilen konu ile ilgili olmasa da PDF dosyalarının otomatik doküman yapısı çıkarımı, matematiksel dokümanların analizi, nesne seviyesinde analiz, mantıksal yapı keşfi, taranmış PDF dokümanlarının düzen analizi, taranmış sağlık hizmeti belgelerinden bilgi çıkarma ile ilgili farklı çalışmalar vardır. Araştırmacılar tarafından PDF içindeki nesneler (metinler, görseller vs.) otomatik olarak çıkartılmış elde edilen veriler satır, paragraf ve daha üst seviye mantıksal yapılara gruplandırılmıştır. Bu şekilde yapısal olmayan verilerden mantıksal temsillerin saptanması genel olarak "doküman anlama / yorumlama" şeklinde literatürde yer bulmuştur.

Doküman anlama genellikle taranmış dokümanlar / görüntüler üzerinde yapılmaktadır [7, 8]. Proje kapsamında, yapılan çalışmalardaki gibi biz de özgeçmiş PDF dokümanlarından isim, soy isim (kişisel bilgiler), iletişim bilgileri, eğitim durumu, iş tecrübesi deneyimler, referanslar, özel zevkler gibi metinsel nesneler ile kişi görüntüsü gibi görsel nesnelerin doküman içindeki konumlarıyla (düzen) tespit edilmesi ve XML formatında ilgili özgeçmişin ifade edilmesi gerçekleştirildi. Mohemad ve arkadaşları [2] Microsoft Word 2007 ortamında oluşturulmuş PDF belgelerinin otomatik doküman yapısı çıkarımını, paragraf ve tablo (ya da çizelge halinde olan) yapılara odaklanarak, Java tabanlı ortamda, JPEDAL isimli java kütüphanesini kullanarak, sezgisel, kural tabanlı ve önceden tanımlanmış göstergeler şeklinde 3 farklı strateji izleyerek gerçekleştirmişlerdir. Hassan [3] ise görsel prensiplere göre aşağıdan yukarıya (bottom-up) bir kümeleme algoritması önermiştir. Chao ve arkadaşı [6] PDF belgesindeki mantıksal bileşenleri tanımlayan teknikler geliştirmiş ve bu mantıksal bileşenlerden anahatlar, stil özellikleri ve içerik çıkartıp sonucu XML formatında ifade etmişlerdir. Liu ve arkadaşları [4] PDF dokümanlarından metin çıkarma araçlarının ortak karşılaştığı metin sırası probleminin çözümü için iki algoritma önermiş ve algoritmaların performansını karşılaştırmışlardır. Gabdulkhakova ve arkadaşı [9] grafik operatörler tarafından dijital PDF belgelerinde çizilen içeriği analiz etmek için nesne tabanlı bir yöntem sunmuşlardır. Baker ve arkadaşları [10] matematiksel metinlerin belge analizi için biri yapısal tanıma için sanal bir bağlantı ağı ile birlikte karakter tanıma (OCR) yaklaşımı ve diğeri düzlem, ifade yapılarını çıkarmak için iki aşamalı ayrıştırıcı ile doğrudan PDF dosyasının sembol bilgisini kullanan iki aşama önermişlerdir.

İlgilenilen bir diğer konu da PDF ve XML'in birbirlerine karşılıklı olarak dönüştürülebilmesidir. PDF dokümanlarından XML dokümanlarının elde edilmesindeki amaç indeksleme ve geri getirim yoluyla dokümanlar üzerinde yapılacak bir arama için arama uzayını (search space) daraltmaktır. Bu tür çalışmalar literatürde "belge özetleme-söylem çıkarımı" olarak geçmektedir. Constantin ve arkadaşları [11] bilimsel PDF dokümanları içinde yer alan başlık, yazarlar, özet, yazar dipnot bilgisi, ana metin, alt başlıklar ve metinleri, resim, tablo, referanslar, bibliyografik kısım gibi 18 tane mantıksal elementi çıkarıp XML tag'ları şekline dönüştürebilen PDFX mimarisini önermişlerdir. Daha sonra ground-truth bilgileri var

olan veri setleri ile yaptıkları çalışmayı test etmişlerdir. Biz de özgeçmişlerde yer alan yukarıda bahsedilen alanları çıkarıp XML tag'lerine dönüştüren, elde edilen XML şablonları üzerinde edit işlemlerine izin veren daha sonra kullanıcı bilgilerine göre yeniden özgeçmiş PDF dosyalarının oluşturmasını sağlayan bir özgeçmiş tasarımcısı geliştirdik.

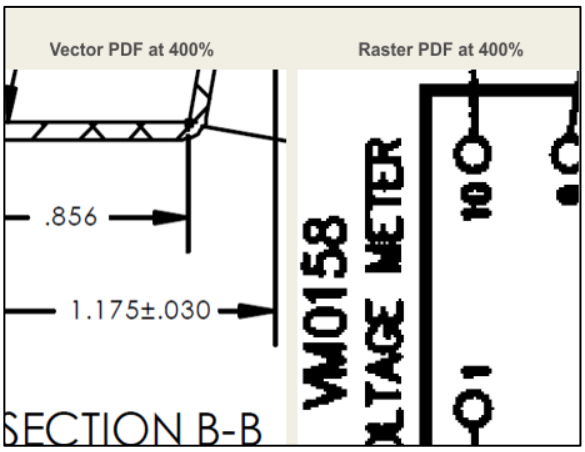
Bölüm 1'in başlarında PDF formatının kısaca tarihçesi ve ne olduğu hakkında ufak bilgi verildi ve Bölüm 1.1'den başlayarak PDF doküman bileşenleri (PDF nesneleri, Dosya yapısı) hakkında detaylı bilgi sunuldu. Bölüm 1.2'de örnek bir PDF dosyası üzerinden Bölüm 1.1'de anlatılan PDF doküman bileşenleri, somut örnekler şeklinde açıklandı. Bölüm 1.3'te de PDF'ten XML'e dönüşüm hakkında literatürde yapılan bazı çalışmalar ve bu tez çalışmasında izlenen PDF-XML dönüşüm stratejisi hakkında bilgi verildi.

Bölüm 2'de genel olarak tez kapsamında faydalandığımız teknolojiler, CV şablonu çıkarmada izlenilen yöntem ve algoritmalar, PDF'ten elde edilen CV şablonunun XML'e dönüştürülme işlemleri hakkında adım adım ve detaylı bilgi verildi.

1. GENEL BİLGİLER

Taşınabilir Belge Biçimi (PDF), Adobe tarafından 1990'larda uygulama yazılımı, donanım ve işletim sistemlerinden bağımsız olarak metin biçimlendirme ve görüntüler de dahil olmak üzere belgeleri sunmak için geliştirilmiş bir dosya biçimidir [12].

PDF dokümanları, oluşturulduğu içerik yapısına göre hem vektör hem de raster içerik barındırabilmektedir. Örneğin vektörel çizim yapılabilen bir uygulamada içeriğinizi PDF formatına çevirdiğiniz zaman PDF dokümanı da vektörel özelliğe sahip olacaktır. Fakat taranmış ya da herhangi bir şekilde oluşturulmuş ama raster özelliğe sahip olan bir içeriği PDF formatına dönüştürdüğünüzde, oluşan PDF formatındaki dosyanız da raster özelliğe sahip olacaktır. Vektör veriler konumu bilinen ve koordinat bilgisine sahip verilerdir. Raster veriler ise hücrelere bağlı olarak pikseller ile temsil edilen ve her hücrede bir renk değerinin depolandığı verilerdir [13]. Bu nedenle vektör PDF dokümanları en iyi şekilde veri çıkarma yoluyla dönüştürülür ve bu işlem doğru ve hassastır. Raster PDF dokümanları ise sadece pixel renk bilgisine sahip olduğundan renk bilgisi haricinde çıkartılabilecek bir veri bilgisi bulunmamaktadır [14]. Bu çalışmada şablon çıkarımı vektörel içerikli, PDF formatlı CV dokümanları üzerinden gerçekleştirildi.



Şekil 1.1. Vektör ve Raster PDF örneği [14]

Şekil 1.1'de %400 zum yapılmış vektör ve raster PDF dokümanları arasındaki kalite farkı gösterilmektedir.

PDF, 2008 yılında ISO 32000 olarak standardize edildi ve artık uygulanması için herhangi bir telif hakkı gerektirmiyor [12].

PDF dosyaları ilginçtir. Bir PDF dosyasını Not Defteri gibi bir metin düzenleyicisinde açarsanız, içerik önemsiz gibi görünebilir ve muhtemelen çok ilginç olmayabilir; ancak PDF dosyalarının belirli bir deseni izlediğini anladıktan sonra biraz daha mantıklı gelmeye başlayacaktır [15].

1.1. PDF Doküman Bileşenleri

PDF sözdizimi dört ana bileşenden oluşur [16]:

1. Objects (Nesneler)
2. File Structure (Dosya Yapısı)
3. Document Structure (Belge Yapısı)
4. Content Stream (İçerik)

1.1.1. PDF nesneleri (PDF objects)

Bir PDF dosyası esas olarak sekiz tür nesneden oluşur [16]:

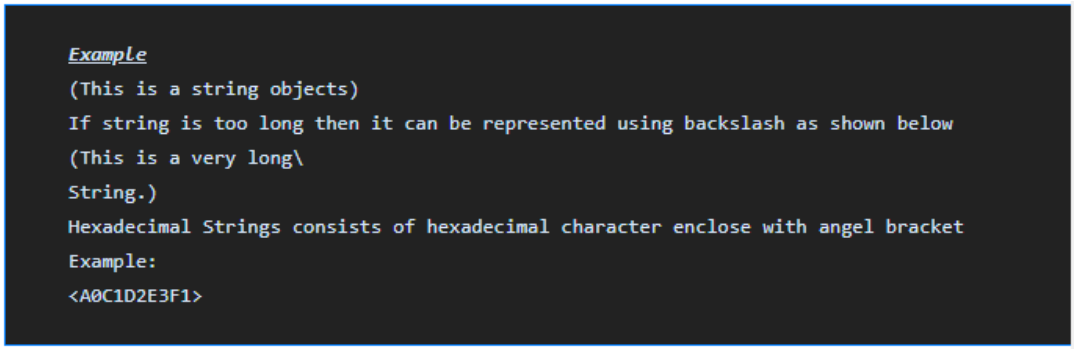
1. Boolean değerler (true veya false)
2. Numbers (Sayılar, integer ve real)
3. Strings (Karakter dizileri)
4. Names
5. Arrays (Diziler, sıralı nesne koleksiyonları)
6. Dictionaries (Sözlükler, isimlerle indekslenen nesne koleksiyonları)
7. Streams (Genellikle büyük miktarda veri içerirler)
8. Null object (Null anahtar sözcüğü ile gösterilen null nesnesi)

1.1.1.1. Karakter dizileri (strings)

Karakter dizileri iki şekilde temsil edilebilir [16]:

- Literal Strings
- Hexadecimal Strings (Onaltılı karakter dizileri)

Literal Strings ler açılış ve kapanış parantezleri arasında herhangi bir sayıda karakterden oluşur.

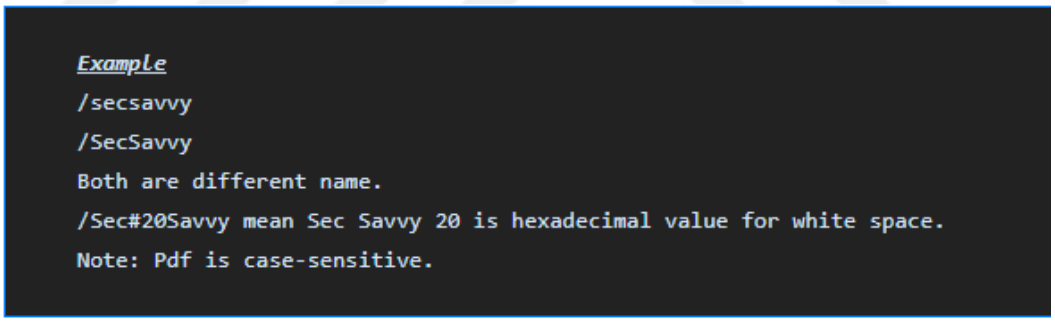


Şekil 1.2. Literal ve Hexadecimal String örneği [16]

Şekil 1.2'de her onaltılık çift bir bayt dizeyi tanımlar.

1.1.1.2. Isimler (names)

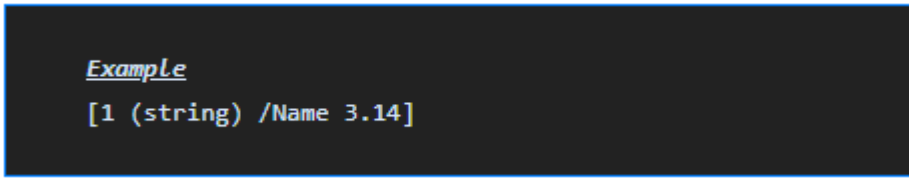
Bir name nesnesi, karakter dizisiyle benzersiz bir şekilde tanımlanır. Şekil 1.3'te bölme işareti (/) ile name nesnesi tanımlama örneği gösterilmektedir.



Şekil 1.3. Isim nesne örneği [16]

1.1.1.3. Dizi (array)

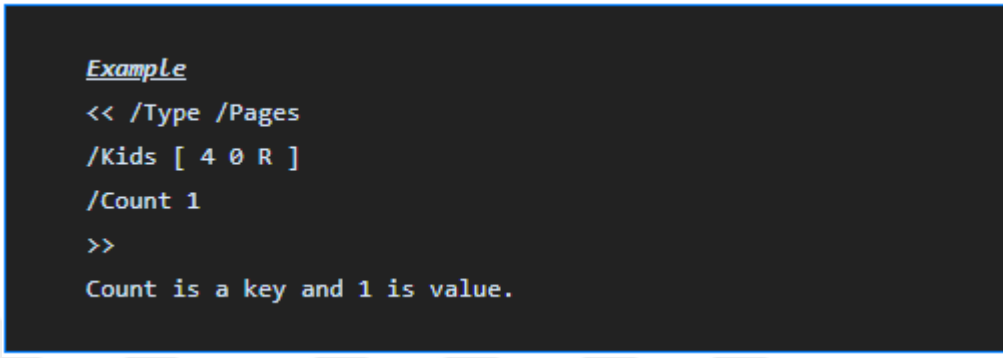
Array nesnesi, nesnelerin koleksiyonudur. PDF dizi nesnesi heterojen olabilir. Şekil 1.4'te köşeli parantez ile array nesnesi tanımlama örneği gösterilmektedir.



Şekil 1.4. Dizi nesne örneği [16]

1.1.1.4. Sözlük (dictionary)

Sözlük nesnesi, nesne çiftlerinden oluşur. İlk öğe anahtar, ikincisi değerdir. Anahtar Şekil 1.3'te gösterilen bir name nesnesi olmalıdır. Dictionary nesnesi Şekil 1.5'te gösterildiği gibi açılı parantez içerisinde key-value çiftler dizisi şeklinde yazılır [16].

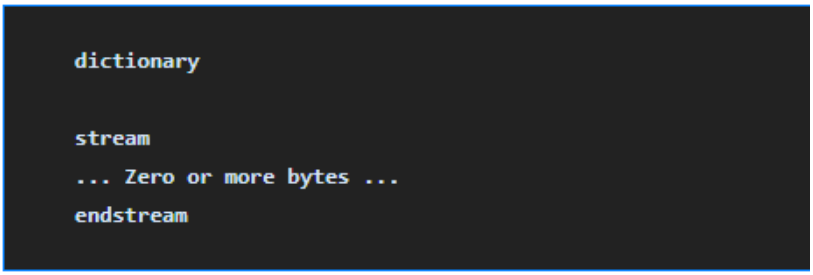


Şekil 1.5. Sözlük nesne örneği [16]

1.1.1.5. Streams

Stream nesnesi, dize nesnesi gibi, bir bayt dizisidir. Stream sınırsız uzunlukta olabilirken, bir dize bir uygulama sınırına tabidir. Bu nedenle, resimler ve sayfa açıklamaları gibi potansiyel olarak büyük miktarda veriye sahip nesneler stream olarak gösterilir [16].

Bir stream Şekil 1.6'da gösterildiği gibi, dictionary ardından stream ve endstream anahtar kelimeleri arasında sıfır veya daha fazla bayttan oluşur.



Şekil 1.6. Stream nesne örneği [16]

1.1.1.6. Dolaylı olan nesneler (indirect ones)

Nesneler, diğer nesneler tarafından refere edilmiş şekilde etiketlenebilir. Etiketli bir nesneye dolaylı nesne denir. Şekil 1.7'de detaylı örnek gösterilmiştir.

```
Example
Consider this object
obj and endobj is a keyword.

10 0 obj
(SecSavvy String)
endobj

This object defined a string of object number 10.
This object can be referred in a file by indirect reference as
10 0 R
```

Şekil 1.7. Indirect nesne örneği [16]

1.1.1.7. Filtreler (filters)

Filtre, bir stream spesifikasyonunun isteğe bağlı bir parçasıdır ve stream deki verilerin kullanılmadan önce kodunun nasıl çözülmesi gerektiğini gösterir. Örneğin, bir stream de ASCIIHexDecode filtresi varsa, stream deki verileri okuyan bir uygulama stream deki ASCII onaltılık kodlu verileri ikili verilere dönüştürür [16].

LZW ve ASCII base-85 kodlaması kullanılarak kodlanan veriler için (bu sırayla) akış sözlüğünde aşağıdaki giriş kullanılarak kod çözülebilir:

/Filter [/ASCII85Decode /LZWDecode]

Şekil 1.8'de yukarıda bahsedilen filter kodunun kullanımına dair örnek içerik çıktısı verilmiştir.

```
Example
1 0 obj
<< /Length 534 /Filter [ /ASCII85Decode /LZWDecode ]>>

stream

J..)6T`?p&<!J9%_[umg"B7/Z7KNXbN'S+,*Q/&"OLT'FLIDK#!n`$"
<Atdi`\Vn%b%)&'cA*VnK\CJY(sF>c!Jnl@RM]WM;jjH6Gnc75idkL5]+cPZKEBPWdR>FF(kj1_R%W_d&/jS!;iuad7h?[L-F$+]]0A3Ck*$I0KZ?;
<)CJtqi65XbVc3\n5ua:Q/=0$W<#N3U;H,MQKqfg1?:lUpR;6oN[C2E4ZNr8Udn.'p+?#X+1>0Kuk$bCDF/(3fL5]Oq)^kJZ!C2H1'TO]Rl?
Q:&<5&iP!$Rq;BXRecDN[IJB`,)o8XJOSJ9sDS]hQ;Rj@!ND)bD_q&C\g:inYC%)&u#:u,M6Bm%IY!Kb1+:aAa$S`ViJgllb8<W9k6Yl
9A jX*al>iG1p&i;eVoK&juJHs9%;Xomop5KatWRT JQ#qYuL,JD?M$0QP)lKn06l1apKDC@\qJ4B!!
(5m+j.7F790m(Vj88l8Q:_CZ(Gm1%X\N1&u!FKHMB~>

endstream
endobj
```

Şekil 1.8. Filter nesne örneği [16]

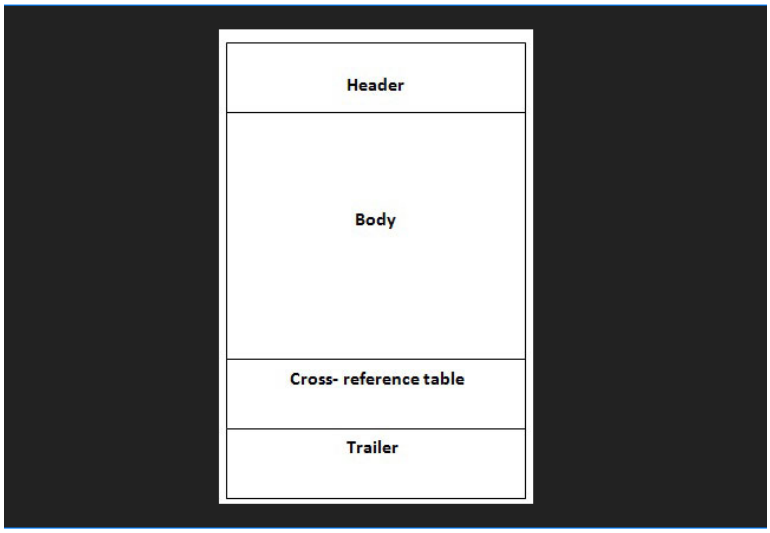
Aşağıda standart filtrelerin listesi verilmiştir:

- ASCIIHexDecode
- ASCII85Decode
- LZWDecode
- FlateDecode
- RunLengthDecode
- CCITTFaxDecode
- JBIG2Decode
- DCTDecode
- JPXDecode
- Crypt

1.1.2. Dosya yapısı (file structure)

PDF dosyası 4 ana öğeden oluşur [16]:

- PDF spesifikasyonunu tanımlayan PDF başlığı (PDF header)
- Dosyada bulunan, belgeyi oluşturan nesneleri içeren bir gövde (body)
- Dosyadaki dolaylı nesneler hakkında bilgi içeren bir çapraz referans tablosu (cross-reference table)
- Çapraz referans tablosunun ve dosyanın gövdesi içindeki bazı özel nesnelerin konumunu veren bir römork (trailer)



Şekil 1.9. PDF dosya yapısı [16]

Şekil 1.9'da PDF dosya yapısının yukardan aşağıya doğru hangi elemanlardan oluştuğu görseli sunulmuştur.

1.1.2.1. Dosya başlığı (file header)

Bu, PDF dosyasının PDF spesifikasyon sürümünü belirtir. % PDF- ve ardından 1.N sürüm numarası gelir; burada N, 0 ve 7 arasında bir rakamdır. Daha önce de belirtildiği gibi, bu aslında PDF sürümünü belirtmek için kullanılan bir yorumdur [15].

1.1.2.2. Dosya gövdesi (file body)

Dosya gövdesi dolaylı nesnelerden oluşur. Bu nesneler, PDF'yi görüntülerken kullanılan metni ve diğer ayrıntıları (yazı tipi türü gibi) temsil eder. Sürüm 1.5 itibariyle, gövde stream nesnesini de içerebilir [15].

1.1.2.3. Çapraz referans tablosu (cross reference table)

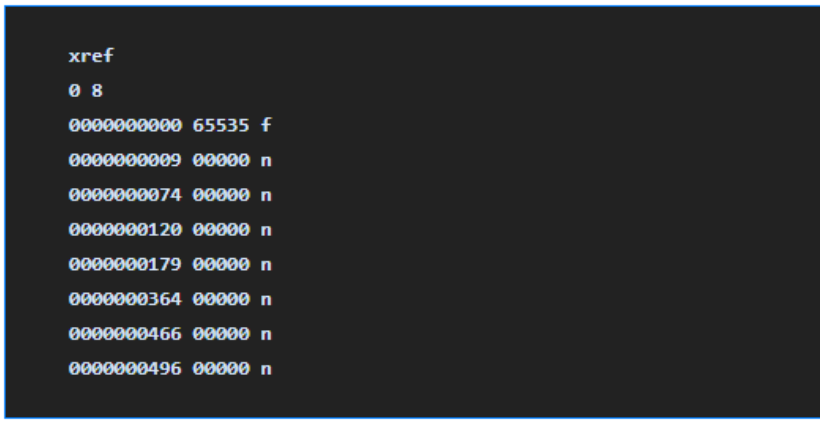
Xref tabloları original PDF dosya spesifikasyonunun bir parçasıdır ve PDF dosya formatına esnekliğini veren ve çok kullanışlı özelliklerden biridir.

Çapraz referans tablosu, dosya içindeki dolaylı nesnelere rasgele erişime izin veren bilgiler içerir, böylece tüm dosyanın belirli bir nesneyi bulmak için okunması gerekmez. Tablo, her dolaylı nesne için, dosyanın gövdesindeki o nesnenin konumunu belirten tek satırlık bir girdi içerir [16].

Her çapraz referans bölümü, xref anahtar sözcüğünü içeren bir satırla başlar. Bu satırın ardından, herhangi bir satırda görünebilecek bir veya daha fazla çapraz referans alt bölümü verilmiştir [16].

Her çapraz referans alt bölümü, bitişik nesne sayıları aralığı için girişler içerir. Alt bölüm, boşlukla ayrılmış iki sayı içeren bir satırla başlar, bu alt bölümdeki ilk nesnenin nesne numarası ve alt bölümdeki giriş sayısı. Örneğin, satır 0 8, ard arda 0'dan 8'e kadar numaralandırılmış beş nesne içeren bir alt bölüm sunar.

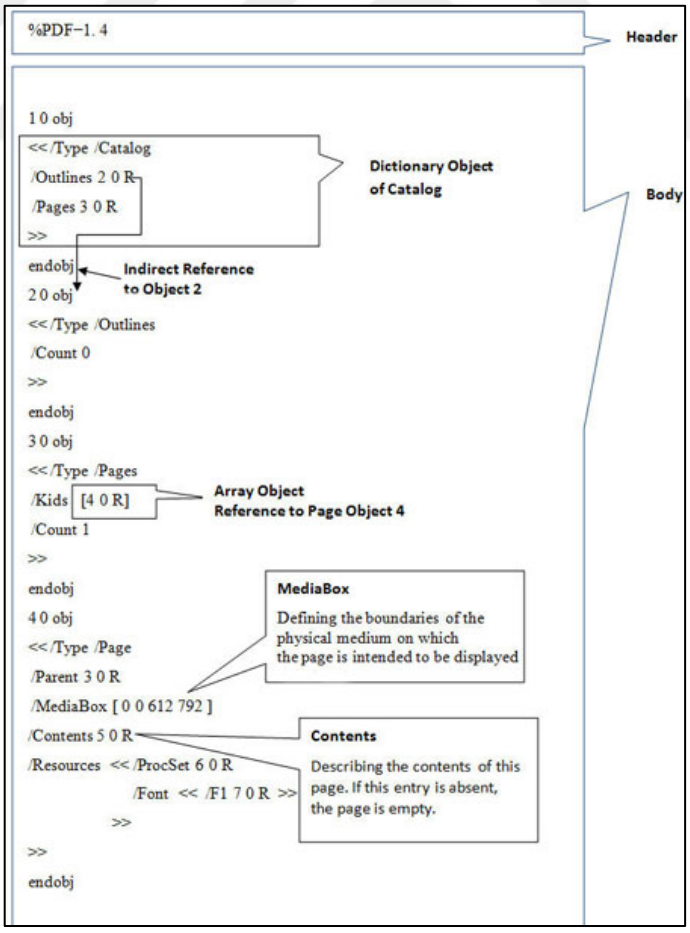
Bir PDF okuyucu bu değerleri okur ve daha sonra nesneleri sadece gerektiğinde yükler. Tüm dosyayı ayrıştırmaya veya yüklemeye gerek yoktur.



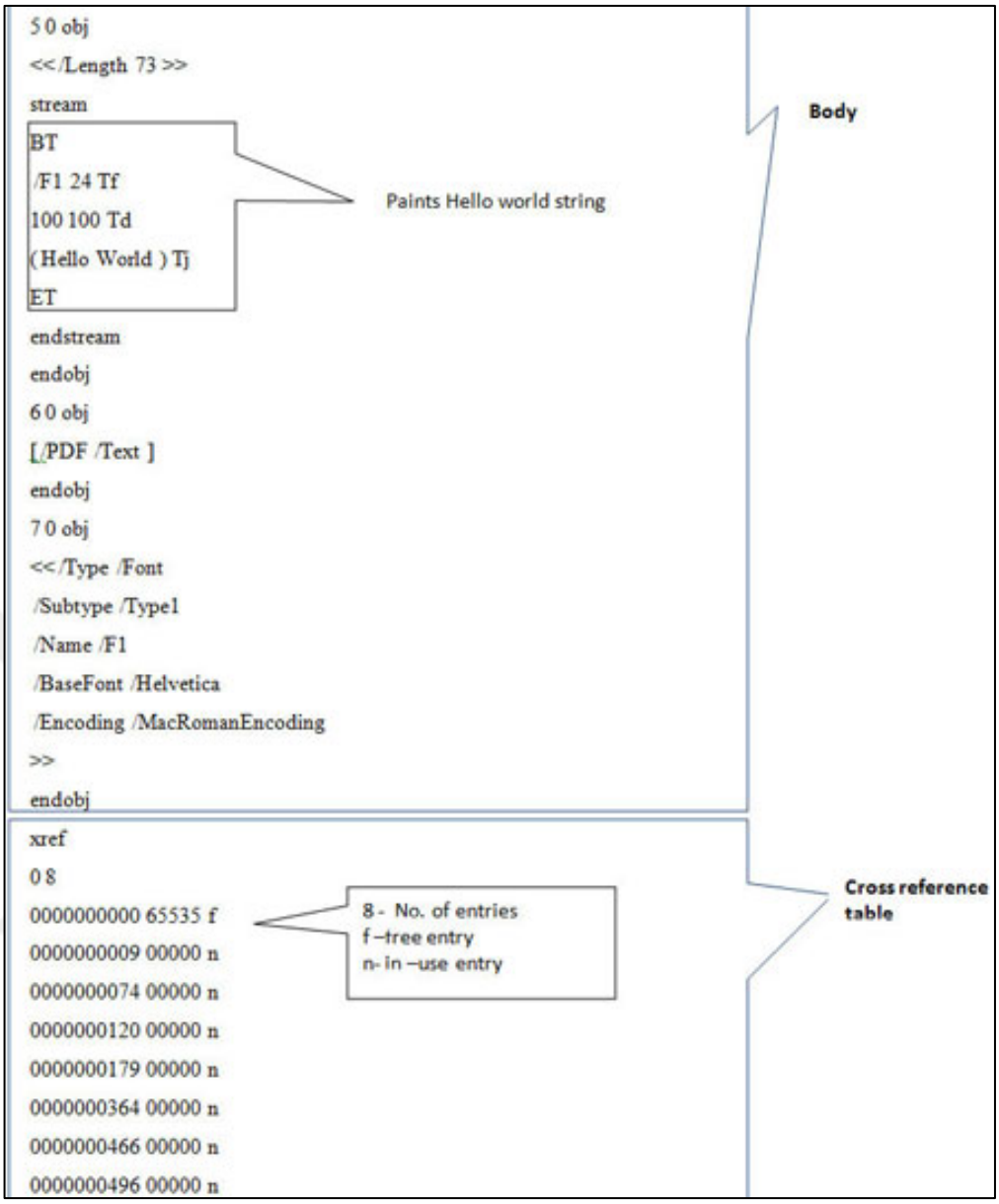
Şekil 1.10. Xref tablo örneği [16]

Şekil 1.10'daki 0000000009 kullanım durumunda 10 haneli bayt uzaklığıdır ve dosyanın başlangıcından nesnenin başına kadar olan bayt sayısını verir.

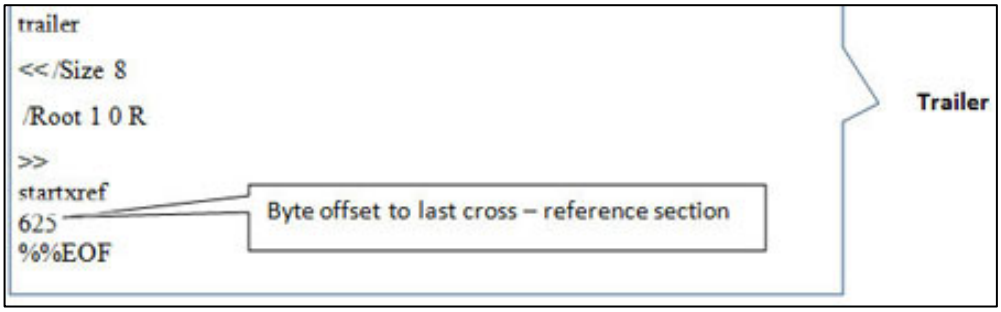
0000000000, serbest giriş durumunda bir sonraki boş nesnenin 10 basamaklı nesne numarasıdır.



Şekil 1.11. Basit Merhaba Dünya PDF örneği 1 [16]



Şekil 1.12. Basit Merhaba Dünya PDF örneği 2 [16]



Şekil 1.13. Basit Merhaba Dünya PDF örneği 3 [16]

Şekil 1.11, Şekil 1.12 ve Şekil 1.13'te PDF belgesinin yukarıda bahsedilen farklı bölümleri gösterilmiştir.

1.2. Örnek Bir PDF Dosyasının Detaylı Açıklaması

Şekil 1.14'te örnek bir PDF dosyası içeriği gösterilmektedir. Yazının devamında aşağıdaki örnek PDF dosyası içeriği alt bölümlere ayırarak detaylı bir şekilde açıklanmaya çalışıldı.

```
%PDF-1.7
1 0 obj
  << /Type /Catalog
     /Pages 2 0 R
  >>
endobj

2 0 obj
  << /Type /Pages
     /Kids [3 0 R]
     /Count 1
  >>
endobj

3 0 obj
  << /Type /Page
     /Parent 2 0 R
     /MediaBox [0 0 600 400]
     /Resources << >>
  >>
endobj

xref
0 4
00000000000 65535 f
00000000010 00000 n
00000000069 00000 n
00000000141 00000 n
trailer
  << /Root 1 0 R
     /Size 4
  >>
startxref
249
%%EOF
```

Şekil 1.14. Örnek bir PDF dosyası içeriği [17]

1.2.1. Nesne sözdizimi (object syntax)

Şekil 1.15'te bir object (nesne) gösterilmektedir.

```
1 0 obj
  << /Type /Catalog
     /Pages 2 0 R
  >>
endobj
```

Şekil 1.15. Nesne sözdizimi [17]

- Objenin adı 1, 0 ise sürüm numarasıdır.
- obj ve endobj bir object in başlangıç ve sonunu tanımlar.
- << >> işaretleri de bir dictionary object tanımlar.

Şekil 1.14'teki object adlarının 1, 2, 3 olduğu unutulmamalıdır. Bu numaralar yerine herhangi bir ad tanımlanabilir.

1.2.2. Başlık (header)

% PDF-1.7 satırı başlıktır ve bu dosyanın PDF 1.7 spesifikasyonunu kullandığını tanımlar [17].

1.2.3. Gövde (body)

Başlığın altındaki ve xref satırının üzerindeki içerikler gövdedir. Bir PDF'yi doğru bir şekilde göstermek için, gövdenin daha sonra konuşacağımız Document Structure (Belge Yapısı) tarafından tanımlanan yapıya sahip olması gerekir [17].

1.2.4. Çapraz referans tablosu (cross reference table)

Bu kısım anlaşılması en zor olan kısımdır. Doğru ayarlanmazsa, PDF görüntüleyici hata verir.

Çapraz referans tablosu, gövdede görünen her nesneye hızlı erişim için kullanılır. Bu yüzden her nesneye çapraz referans girişi vermeliyiz [17].

```
xref
0 4
0000000000 65535 f
0000000010 00000 n
0000000069 00000 n
0000000141 00000 n
```

Şekil 1.16. Cross-reference tablo içeriği [17]

Şekil 1.16'da gösterilen cross-reference tablosu xref anahtar kelimesi ve bir EOL ile başlar. Sonra başlangıç objesini gösteren bir satır gelir.

Şekil 1.16'daki başlangıç objesi 0 ve ondan sonraki 4 rakamı PDF gövdesinde ardışık 4 nesneye karşılık gelmektedir.

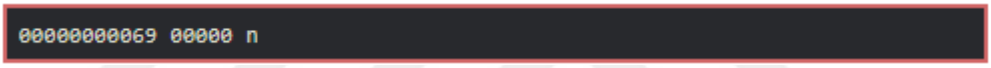
Fakat Şekil 1.14'teki örnek PDF dosyasında 1, 2 ve 3 objeleri yani 3 obje gösterilmekte. 0. obje gövdenin kökü (Belge Kataloğundan farklıdır) olduğu için gösterilmez [17].

Sonrasında birkaç satırlık birden fazla giriş gelmektedir. Şekil 1.17'de bu girişlerin yapısı (biçimi) gösterilmektedir.

Şekil 1.17. Object formatı [17]

- nnnnnnnnnn belgenin başından başlayarak objenin 10 basamaklı bayt uzaklığıdır.
- ggggg mevcut objeyi göstermek için tanımlanmış 5 basamaklı bir nesil numarasıdır. Obje her silindiğinde ve yeniden kullanıldığında, yeni bir nesil numarası tanımlanır.
- giriş tipi, n kullanımda, f kullanımda olmayan anlamına gelir.

Toplamda giriş tam olarak 20 bayt yer kaplar.



Şekil 1.18. Xref tablosundaki 3. girdi [17]

Bizim örneğimizde Şekil 1.18'de gösterilen girdi 3. girdidir yani Şekil 1.19'daki objeyi göstermektedir.

Şekil 1.19. Gövdedeki 3. Obje [17]

Şekil 1.18'deki 69 sayısı objenin ofseti dir ve 00000 sayısı da nesille ilgili herhangi bir özellik kullanılmadığını gösterir. Son bilgi olan n harfi de girdinin kullanımda olduğunu belirtir [17].

1.2.5. Trailer

Trailer bölümü bize PDF belgelerinin genel bilgilerini verir, en az iki girişi olması gereken bir dictionary içermelidir. Şekil 1.20'de gösterilen /Root ve /Size gibi.

Şekil 1.20. Trailer bölümü [17]

/Root, gövdenin Kataloğuna atıfta bulunur (yazının devamındaki bir sonraki bölüm). /Size, dosyanın çapraz başvuru tablosundaki toplam girdi sayısını ifade eder.

startxref, satırından sonra bir sayı satır bilgisi gelmektedir. Bu sayı çapraz referans tablosunun başlangıç ofsetini (xref anahtar kelimesinin uzaklığını) gösterir.

%% EOF dosyanın sonunu gösterir.

Trailer bölümünün temel amacı, dosyayı görüntüleyenin, dosyayı alttan okuyabilmesi ve:

- startxref bilgisi ile çapraz referans tablosunun ofsetini öğrenebilmesi
- trailer objesindeki /Root girdisiyle kök nesneyi bulabilmesidir.

Ardından dosyayı görüntüleyen, çapraz referans tablosu yardımıyla 0000000010 konumunda kök nesnenin gerçek içeriğini (bizim örneğimizde 1 0 R) bulabilecektir. Kök, /Page 2 0 R'de yazıldığı gibi nesne (2 0 R) yi içerir. Böylece dosyayı görüntüleyen, çapraz referans tablosu yardımıyla goto offset 0000000069 bilgisini bulacaktır. Bu şekilde devam edilerek tüm PDF dosyası okunup görüntülenebilir [17].

1.2.6. Belge kataloğu (document catalog)

Şekil 1.21. Belge Kataloğu [17]

Şekil 1.21'de gösterilen Belge Kataloğu'nda:

- /Type /Catalog bilgisi belirtilmelidir.
- /Pages girişi belitilmelidir.

1.2.7. Sayfalar (pages)

Sayfalar düğümü, sayfa ağacının kök düğümüdür. Şekil 1.22'de sayfa ağacı hiyerarşisine bir örnek verilmiştir.

```
2 0 obj
  << /Type /Pages
     /Kids [ 4 0 R
             10 0 R
             24 0 R]
     /Count 3
  >>
endobj

4 0 obj
  << /Type /Page
  ... Additional entries describing the attributes of this page ...
  >>
endobj

10 0 obj
  << /Type /Page
     ... Additional entries describing the attributes of this page ...
  >>
endobj

24 0 obj
  << /Type /Page
     ... Additional entries describing the attributes of this page ...
  >>
endobj
```

Şekil 1.22. Sayfalar düğümü [17]

- /Pages'te /Type bilgisi belirtilmelidir.
- Kök düğüm dışında /Parent bilgisi set edilmelidir.
- /Kids dizisi alt sayfaları belirtecek şekilde ayarlanmalıdır.
- /Count yaprak düğümlerinin sayısını belirtecek şekilde ayarlanmalıdır.

1.2.8. Sayfa nesnesi (page object)

```
3 0 obj
  << /Type /Page
     /Parent 2 0 R
     /MediaBox [0 0 600 400]
     /Resources << >>
  >>
endobj
```

Şekil 1.23. Sayfa nesnesi [17]

Şekil 1.23’te gösterilen Page objesinde:

- /Type bilgisi belirtilmelidir.
- /Parent bilgisi bir üst nesneyi belirtecek şekilde ayarlanmalıdır.
- /MediaBox ortam içeriğini depolamak için sayfada bulunan dikdörtgen bir yapıdır.
- /Resources bu sayfa için gereken kaynakları (örn. Yazı tipleri) içerir.

1.3. PDF’ten XML’e Dönüşüm

Araştırmalarımıza göre özgeçmiş dokümanları için yapılmış PDF dosyasından XML’e dönüşüm yapan bir çalışmaya rastlamadık. Ancak akademik çalışmalar (bildiri, makale) mevcuttur. Şekil 1.24’te ise önerilen yaklaşımla elde edilmiş bir PDF dosyası ve ona ait XML dosyası resmedilmiştir.

Şekil 1.24. a) pdfx girdi ve b) XML çıktı şablonu

Biz de benzer şekilde özgeçmiş dokümanındaki şablon yapısını çıkartıp tüm stil bilgilerinin içinde bulunduğu bir XML dosyası ürettik. Son kullanıcının beğenmiş olduğu tüm özgeçmiş PDF’leri için farklı XML şablonları üretilebilmektedir. Oluşturulan XML şablonları yardımıyla kullanıcının kendine ait CV bilgileri otomatik olarak hazır şablon formatına giydirilerek kişiye özel CV örnekleri oluşturulabilmektedir. Farklı amaçlar için dokümanlardan metin verilerinin, görüntülerin, tablo veya formların çıkarılması söz konusu ise de format şablonlarının oluşturulması ve spesifik olarak CV'ler üzerinde uygulanmasının yapılması ile ilgili olarak bir çalışmaya rastlanmamıştır.

2. MATERYAL VE METOT

PDF formatındaki CV dosyalarının tasarım şablonlarının otomatik olarak tanınması ve tasarımın farklı içeriklere uyarlanması gerçekleştirildi. Java kütüphanesi olan iText-pdf kullanılarak PDF içeriği tarandı ve şablon oluşturuldu.

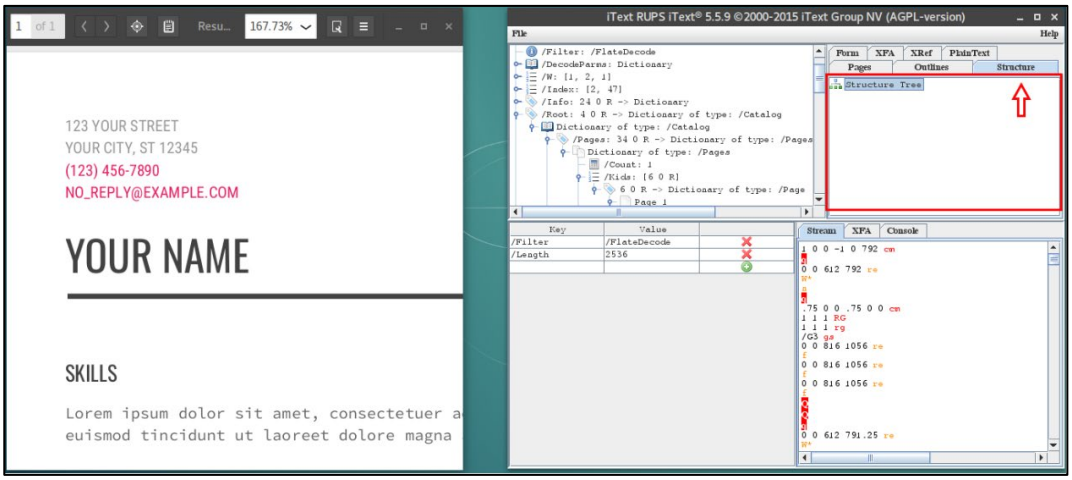
Proje kapsamında vektörel yapıya sahip PDF dokümanları üzerinde çalışıldı. Bu çalışmada kullanılan yöntem raster yapılı PDF dokümanlarında çalışmamaktadır. Raster yapılı PDF dokümanlarında sadece pixel renk bilgilerine ulaşılabildiği için OCR gibi teknikler dışında bir yöntemle içerik çıkarmak mümkün olmamaktadır.

Proje kapsamında iText-pdf isimli kütüphane kullanıldı fakat PDF veya herhangi bir formattaki doküman içeriğini okuyabilmeye yarayan open ya da closed source olan çok sayıda kütüphane bulunmaktadır. Birden çok format yapısını destekleyen ve doküman içeriğini okumaya yarayan kütüphanelerden biri de Apache Tika kütüphanesidir. Apache Tika otomatik olarak doküman formatını tanıyarak o formata göre içerik okumayı gerçekleştirebilmektedir. Apache Tika PDF dokümanlarını parse etmek için özellikle PDF belgeleri ile çalışmak için geliştirilmiş olan Apache PDFBox kütüphanesini kullanmaktadır [18]. İlk olarak, PDF HTML gibi bir biçimlendirme biçimi değildir. Yani, PDF dosyaları farklı yazı tiplerini belirten ekstra dekorasyona sahip düz metin bilgilerini içermemektedir. HTML'den ziyade PostScript'e daha yakın olan bir sayfa açıklama biçimidir. PDF dosyasındaki her sayfa, bir dizi komut içeren bir içerik akışı ile tanımlanır. Bu komutlar yardımıyla belirlenen metin çizilir, geçerli yazı tipi ve renkler değiştirilir ve farklı geometrik şekiller çizilebilir. Genellikle metin, karakter aralığını sağlamak amacıyla küçük parçalar dediğimiz chunk'lara ayrılmaktadır. Böylece, örneğin "Türkiye" metni çizileceği zaman önce "T" chunk'ı çizilir ardından biraz sağa kayıp "ü" harfi ve işlem bu şekilde tüm chunk'lar üzerinde uygulanıp kelime, satır ve paragraf bilgileri oluşturulmaktadır. Bir PDF dosyası herhangi bir sayıda yazı tipi kullanabilir. Bir yazı tipi, gliflerin bir koleksiyonudur - Times-Roman, Helvetica ve Courier'in her birinin, örneğin 'A' harfi için kendi glifleri vardır. Her yazı tipinde ayrıca karakter kodlarından (sayılardan) glif adlarına eşleme olan bir kodlama (encoding) vardır. Bu kodlama bazen ASCII kodlama sistemine

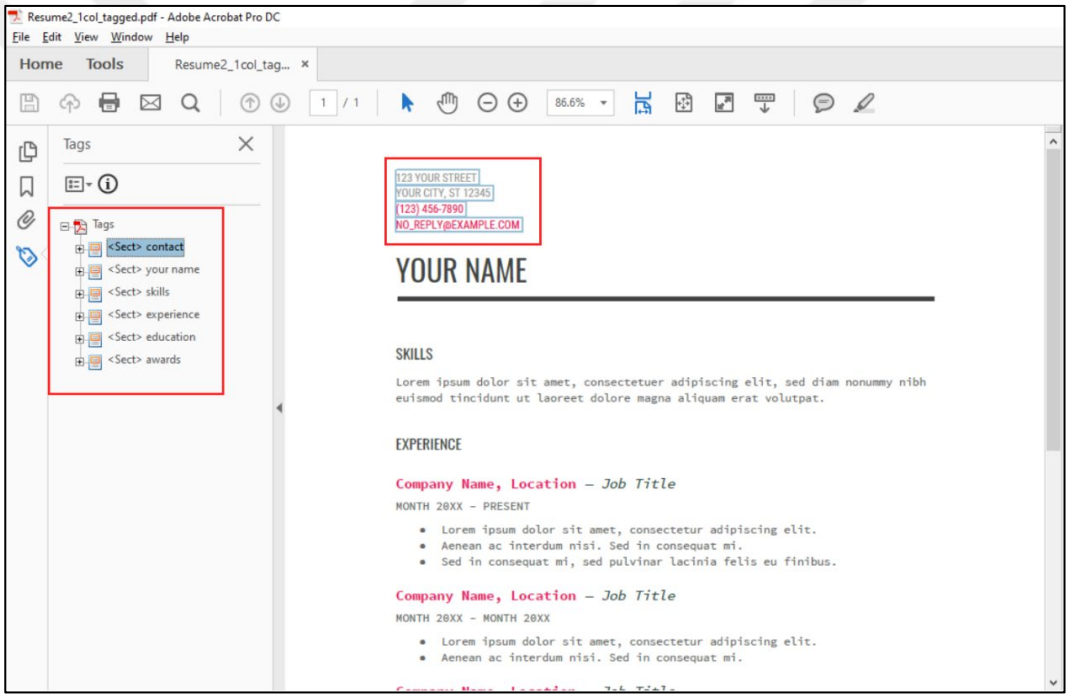
benzemektedir. Örneğin kod 46 ‘nokta’ ile, kod 48 ‘sıfır’ ile, kod 65 ‘A’ ile eşleşmektedir. Bir PDF metin çıkarıcı, yazı tipi kodlamasını kullanarak kodu glif adlarına ve glif adlarını da çıktı kodlamasının belirlediği bilgilere dönüştürür [19]. Apache PDFBox ya da iText gibi PDF içerik okuma kütüphaneleri de bu chunkları tekrar belirli çıkarım stratejileri uygulayarak karakter dizisine birleştirmektedir. Projeye başlamadan önce kütüphane seçim aşamasında Stackoverflow’da ki bir PDFBox ve iText performans karşılaştırması konu başlığı altında PDFBox kütüphanesinin metni glyph işleme mantığıyla işlediğini, iText kütüphanesinin ise yukarda bahsettiğimiz chunk mantığıyla işlediğini ve bu özelliği yardımıyla kütüphanenin gerek duyduğu kaynakları azaltıp PDFBox’a göre daha performanslı çalıştığı bilgisi baz alınarak kütüphane seçildi ve ilerleyen aşamalarda performans ya da diğer konularda kütüphane kaynaklı bir sorun yaşanmadığı için iText kütüphanesi ile devam edildi [20].

Şablonun XML formatında kaydedilmesi ve farklı içeriklere uyarlama aşamasında, kaydedilmiş hazır XML formatındaki şablonların kullanılması öngörüldü. XML formatındaki şablonların otomatik oluşturulabilmesi ve sonradan doğruluğunun test edilebilmesi için XML tanımlandı.

PDF verileri yapısal olmayan verilerdir buna gore bu verilerin tanımlanabilir bir yapısı yoktur. Bu yapıları gereği pdf verilerinin tasarım şablonlarının çıkartılması projede çözulmeye çalışılan problemlerden biridir. PDF verileri etiketleme özelliği kullanarak yapısal hale getirilebilir; fakat genelde çoğu pdf verisi etiketlenmemiş ve yapısal olmayan haldedir. Şekil 2.1’de kırmızı işaretli alanda bir PDF inceleme aracı olan iText RUPS uygulaması ile PDF dokümanının yapısı ağaç görünümü ile gösterilmektedir. Şekil 2.1’de sağ üst köşede kırmızı diktörgen ve ok ile gösterilen yerde PDF’in yapı bilgileri gösterilmektedir. PDF belgeleri tag özelliği yardımıyla yapısal hale getrilebilir. Şekil 2.1’de Structure alanının boş olması PDF dosyasında taglenmiş herhangi bir bilginin bulunmadığını gösterir. Structure bölümünün tam altında da Stream penceresinde aslında bir bayt dizisi olan Stream bilgileri gösterilmektedir. Stream özelliği yardımıyla tek bir ikili Stream içinde çok sayıda PDF nesnesi bir araya getirilebilir. Bu özellik aynı zamanda PDF’i daha küçük boyutlu, potansiyel olarak daha güvenli ve daha hızlı yüklenir hale getirmektedir.

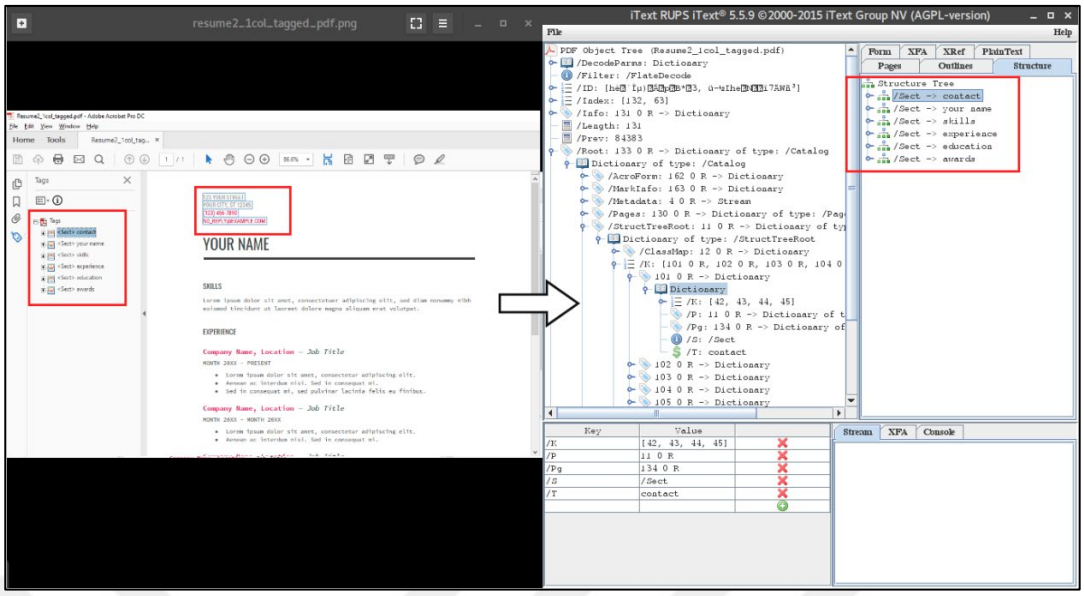


Şekil 2.1. Taglenmemiş PDF (iText RUPS)



Şekil 2.2. Bir kısmı taglenmiş PDF (Adobe Acrobat)

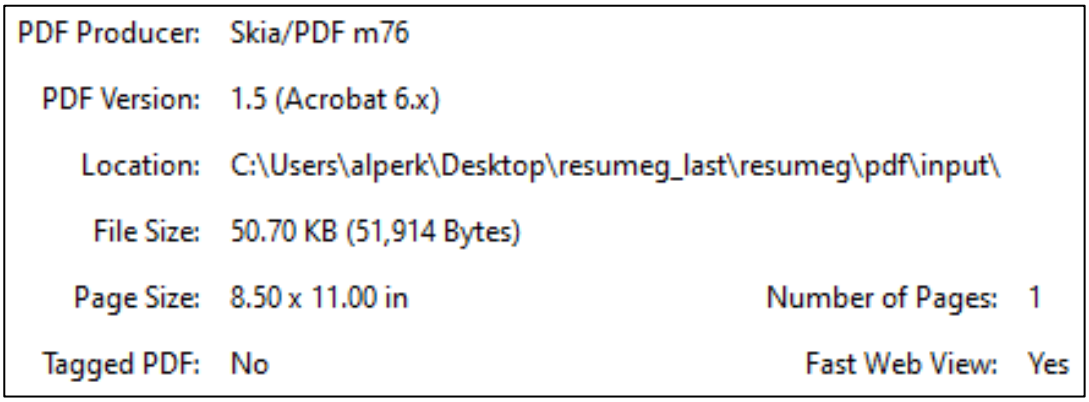
Şekil 2.2'de gösterilen taglenmiş PDF örneği Şekil 2.1'de soldaki PDF örneğinin manuel yolla Adobe Acrobat uygulaması kullanılarak taglendirilmiş halidir. Şekil 2.3'te de aynı PDF örneğinin iText RUPS uygulaması ile açılmış hali bulunmaktadır. PDF tagleme özelliği yardımıyla PDF ten kolay yolla veri çekilmesini istediğiniz bir kısmını tagleyebilir ve tag ismi ile kolayca PDF içerisinden taglenen kısımların metin içeriğini, font tipini, karater boyutunu, font rengini, yazının PDF sayfası üzerindeki konumunu çıkartabilirsiniz. Kolay okunup işlenebilen PDF dokümanları hazırlayabilmek için tagleme özelliği önemlidir.



Şekil 2.3. Bir kısmı taglenmiş PDF (iText RUPS)

Şekil 2.1'de görüldüğü gibi yapısal olmayan PDF dokümanlarında bir yapı tanımlanmadığı için iText RUPS uygulamasının Structure sekmesindeki Structure Tree listesi boştur. Bu tür PDF dokümanları bir yapıya sahip olmadığı için, PDF içeriği baştan sona taranarak, doküman içerisindeki ögeler ve ögeler arasındaki ilişkiler analiz edilerek PDF yapısı oluşturulabilir.

Şekil 2.2 ve Şekil 2.3'te ise Tagged (taglenmiş, yapısal) PDF yapısı görülmektedir. Bu tür PDF dokümanlarında yapı önceden tanımlandığı için, içerik okuma, değiştirme, tanımlanan yapı yardımıyla çok daha kolay bir şekilde gerçekleştirilebilir.



Şekil 2.4. Örnek PDF doküman metadatası

Şekil 2.4'te projede girdi olarak kullanılan örnek bir CV dokümanının metadata bilgileri gösterilmektedir. Projede kullanılan CV örnekleri Google örnekleri olduğu

için PDF Producer bilgisinde Chrome tarayıcısında kullanılan ve PDF arka ucuna sahip bir grafik kütüphanesi olan Skia bulunmaktadır. CV dokümanları PDF 1.5 sürümü kullanılarak oluşturulmuş. Tagged PDF bilgisindeki No bilgisi de dokümanın taglenmiş bir PDF yapısına sahip olmadığı yani Şekil 2.1'de gösterile PDF yapısında olduğu görülebilmektedir. Diğer tüm CV girdileri de Şekil 2.4'de gösterilen PDF yapısına sahip girdileri teşkil etmektedir.

2.1. PDF Şablon Çıkarımı

Şekil 2.7'de bu çalışmanın genel algoritma akış diagramı gösterildi. Algoritmadaki 1. Adım normal herhangi bir text dosyasını okuma işlemi gibi genel ve basit bir işlem olan PDF dokümanını açma işlemini kapsamaktadır. Algoritmanın asıl önemli, detaylı ve geniş adımlarını kapsayan diğer adımlar yazının devamında alt başlıklar açılarak ve açıklayıcı şekiller ile desteklenerek anlatıldı. Özet olarak, ilk önce iText-pdf kütüphanesi yardımıyla PDF dokümanı açıldı. Açılan PDF dokümanının içerisindeki metin öğeleri (bileşenleri) okundu. Okunan metin öğeleri rastgele bir sırada okunabildiği için ve bu bizim işimize yaramadığı için mantıksal okuma sırasına göre sıralandı. Elde edilen karakter ya da sözcük öğeleri kullanılarak satır yapıları oluşturuldu. Satır yapıları birden fazla sütun yapısına sahip PDF dokümanlarında sütun bazında gruplandı. Elde edilen satır yapıları birleştirilerek paragraf yapıları oluşturuldu. Son olarak elde edilen satır ve paragraf yapıları kullanılarak kural tabanlı yaklaşımla PDF şablon yapısı çıkartıldı (oluşturuldu).

Şekil 2.5. TagConent nesne yapısı

Şekil 2.5'te TagContent nesne yapısı class gösterimi ile sunulmuştur. Birinci değişken olan tagName değişkeni String yapıya sahip ve herhangi bir tag'e ait olan key görevini görmektedir. İkinci değişken olan content değişkeni de içerisinde sadece String yapıda bilgi tutabilen Java List yapısına sahip. Örneğin adres, telefon, mail gibi bilgiler

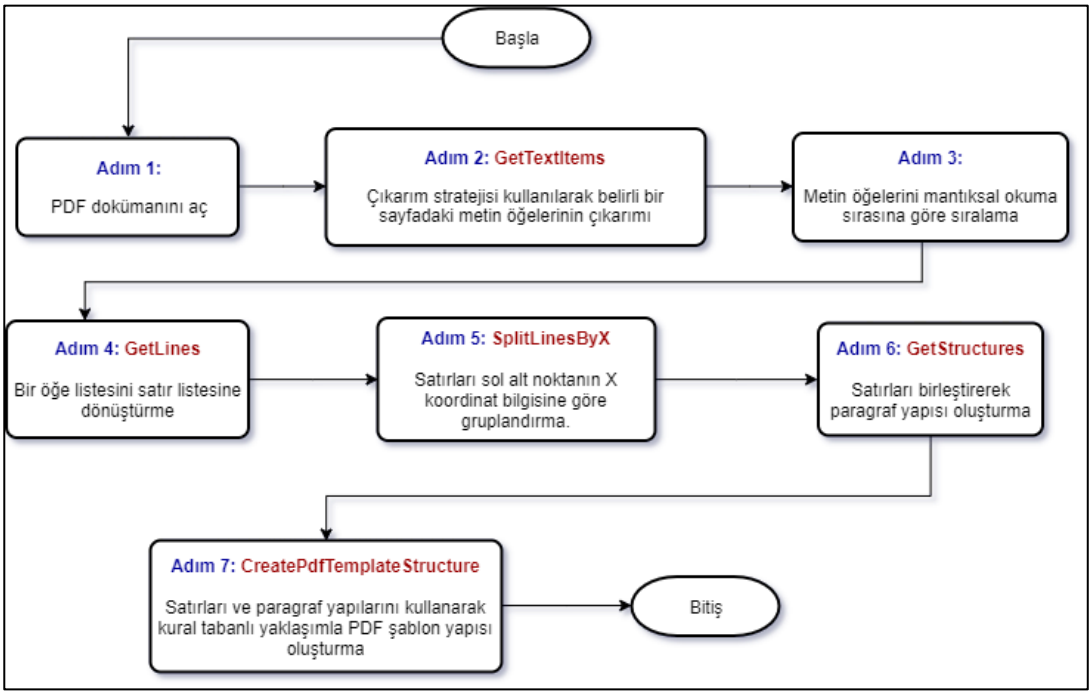
iletişim kategorisine girdiği için ‘contact’ tagName’i ile bir TagContent nesnesi oluşturup content değişkenine de adres, telefon, mail gibi bilgileri topluyoruz.

List<TagContent> tagContents -> String tagName, List<String> tagContent

"name"	"Your Name"
"street"	"123 Your Street"
"city"	"Your City, ST 12345"
"phone"	"(123) 456-7890"
"email"	"no_reply@example.com"
"title"	"Experience", "Education", "Projects", "Skills", "Awards", "Languages"

Şekil 2.6. Örnek TagConent listesi ve içeriği

Kural tabanlı yaklaşımda Şekil 2.5’te gösterilen TagContent nesne yapısındaki Liste değişkeni kullanıldı. Üzerinde çalışılan PDF dosyaların içeriği baz alınarak Şekil 2.6’da gösterildiği gibi TagConent listesine önceden örnek bilgiler girildi. 2.1.6 başlığı altında anlatılan PDF şablon yapısının oluşturulması kısmında şablonun oluşturulması, Şekil 2.6’da gösterilen tagContents listesi ile PDF ten okunan içeriğin eşleştirilmesiyle gerçekleştirildi.



Şekil 2.7. PDF şablon çıkarım algoritması

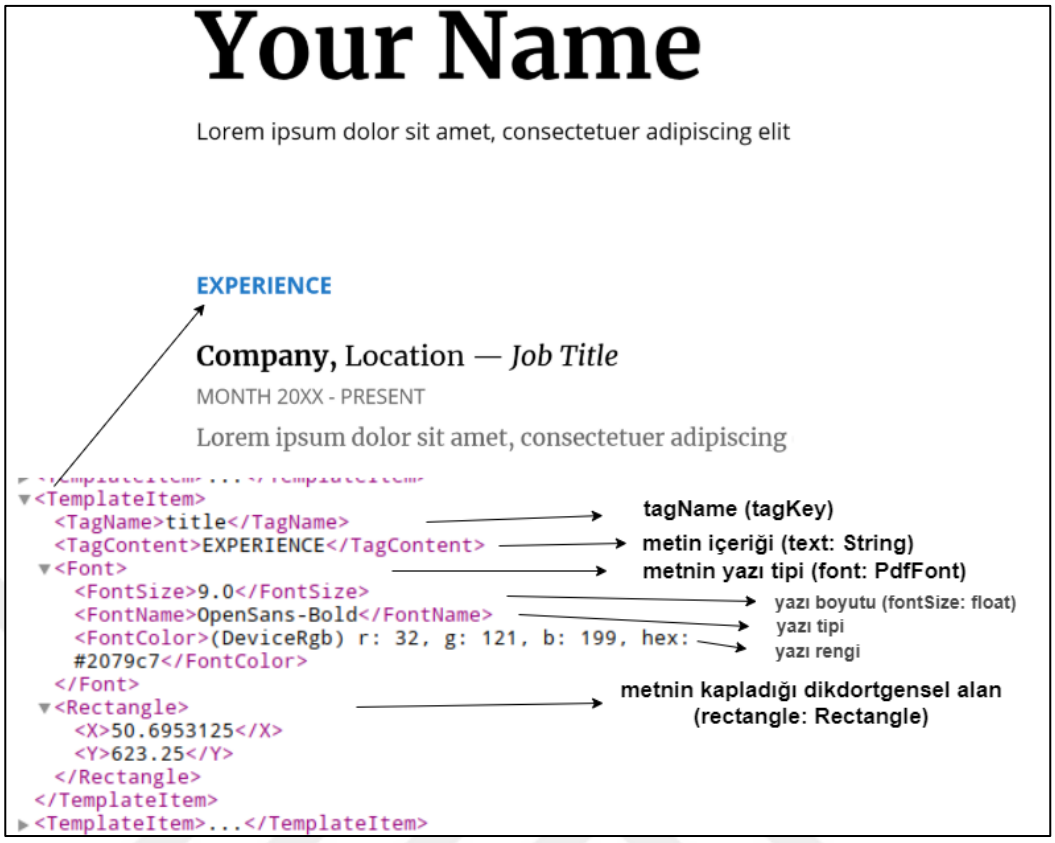
2.1.1. Metin öğelerinin çıkarımı (get text items)

PDF dokümanı parçalanıp küçük anlamlı parçalar elde edildiğinde, bu parçalar genelde bir karaktere karşılık gelmektedir. Dokümanın tamamını ya da bir parçasının şeması çıkartılmak istendiğinde bir karaktere ait bilgiler yeteri kadar anlam ifade etmemektedir. Bu nedenle bu karakterler kelime, cümle ya da paragraf olarak gruplanır. Dokümanı oluşturan bir sayfa karakter karakter parçalanıp satır olarak gruplandığında şema oluşturmak için yeterli bilgiler elde edilebilir.

Bu aşamada iText-pdf kütüphanesi kullanılarak PDF dokümanı metni tarandı. PDF şablonu metin içerisindeki item (öğe) ler ve bunların arasındaki ilişki baz alınarak oluşturulacağı için tüm item ler özellikleriyle beraber taranıp TextItem nesne tipinde oluşturuldu ve List<TextItem> items listesinde tutularak Şekil 2.7'de gösterilen Adım 3'e girdi olarak gönderildi.

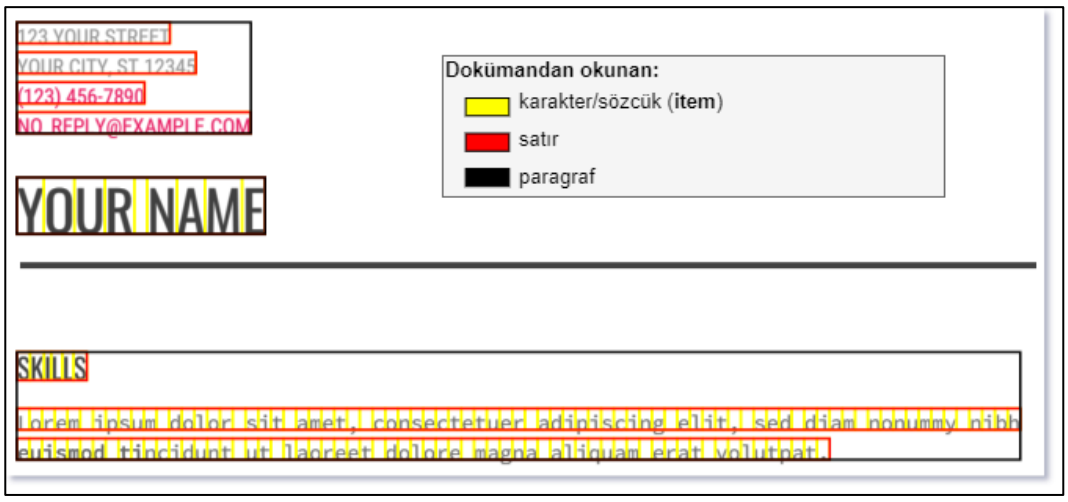
Şekil 2.8. TextItem nesne yapısı

Şekil 2.8'de TextItem nesne yapısı class gösterimi ile sunuldu. Birinci değişken olan text değişkeni String yapıya sahip ve ve PDF içerisinden okunan herhangi bir öğenin metin içeriğinin tutulduğu yerdir. İkinci değişken olan rectangle değişkeni de bizim tanımladığımız custom bir class olan Rectangle class yapısına sahip ve okunan metnin PDF dosyası üzerindeki konum bilgisi bu değişken üzerinde tutulmaktadır. Üçüncü değişken olan font değişkeni iText kütüphanesi içerisinde bulunan PdfFont class yapısına sahip ve PDF'ten okunan metnin yazı tipi bilgisi bu değişken üzerinde tutulmaktadır. Dördüncü değişken olan fontSize değişkeni float yapısında ve bir önceki font değişkeni bilgileri kullanılarak elde edilen yazı tipi boyutu bilgisi bu değişken içerisinde tutulmaktadır. compareTo fonskiyonu da öğeleri okuma sırasına göre sıralama fonksiyonudur.



Şekil 2.9. PDF dosyasından elde edilen format özellikleri

Şekil 2.9'da PDF şablonunun XML formatındaki çıktısı ve XML dosyasındaki taglerin ne tür bilgilere karşılık geldiği Şekil 2.8'teki TextItem nesne yapısı referans alınarak gösterildi. Şekil 2.9'daki EXPERIENCE başlığı örneğinden gidersek, kişiye özel oluşturulacak yeni PDF dosyasında kişinin deneyim bilgileri title tagName'ine bağlı TemplateItem'dan çekilen bilgiler doğrultusunda oluşturulmaktadır.

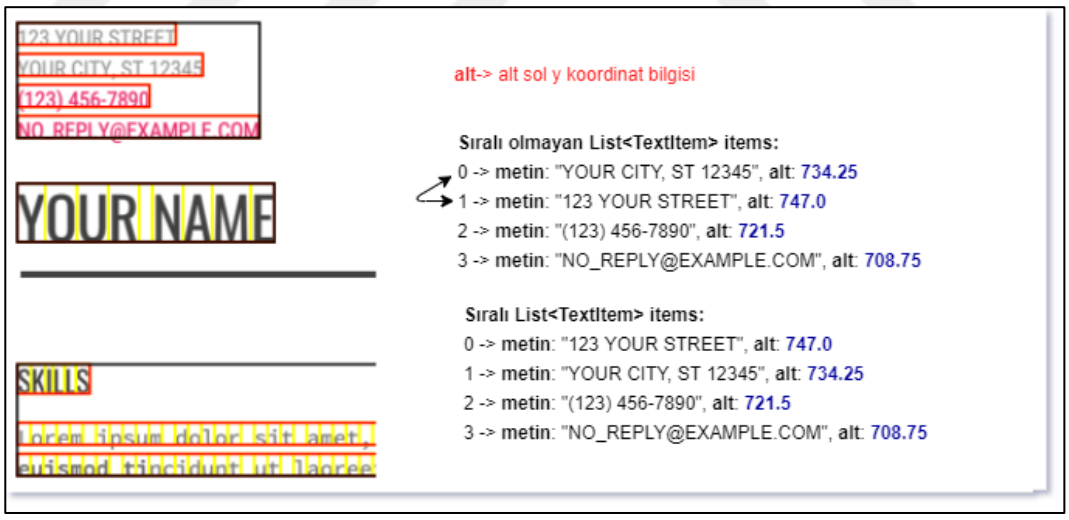


Şekil 2.10. Item, satır ve paragraf bileşenleri

Şekil 2.10'da gösterildiği gibi item olarak adlandırdığımız bileşen PDF yapısına bağlı olarak bazen karakter bazen de bir bütün sözcük halinde, iText-pdf kütüphanesinin hazır bir fonksiyonu kullanılarak okundu. Satır ve paragraf bileşenleri ise, okunan item bileşenlerine Şekil 2.7'deki Adım 4 ve Adım 6'da gösterilen algoritmaların uygulanması sonucu oluşturuldu.

2.1.2. Mantıksal okuma sırasına göre sıralama (sort text items)

iText-pdf kütüphanesi ile PDF içeriği okunurken, PDF dokümanında tanımlı olan Xref tablosundaki bilgilere ve tabloda belirlenen sıraya göre içerik okunuyor. Bu okuma, bir insanın okuması gibi yukarıdan aşağıya doğru bir yönde olmayabiliyor. PDF şablonunu oluşturabilmek için dokümandaki item lerin mantıksal okuma sırasına göre sıralanması önemli. Bu sorunu çözebilmek icin Şekil 2.8'de gosterilen compareTo (TextItem) fonksiyonu tanımlandı. Bu fonksiyon Java'daki Collections.sort() fonksiyonunun TextItem tipindeki bir listeyi sıralarken izleyeceği karşılaştırma ve sıraya dizme kurallarını barındırmaktadır. Aşağıdaki görselde, liste sıralaması yapılırken uygulanan compareTo karşılaştırması detaylı bir şekilde gösterildi.



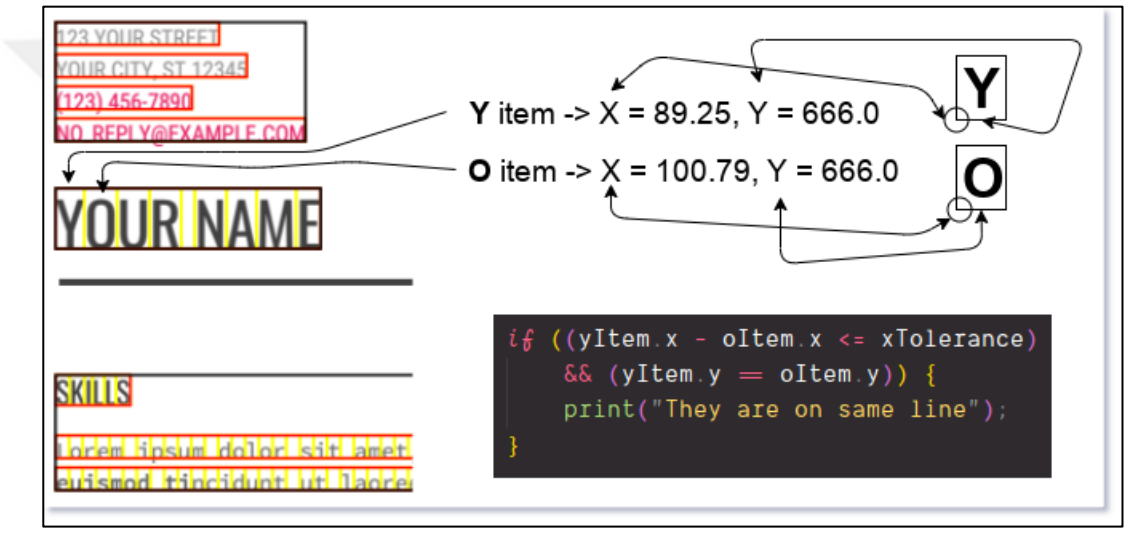
Şekil 2.11. List<TextItem> items, listesinin sıralanması

Şekil 2.11'de gösterildiği gibi sıralanmamış items listesinde 0. Eleman'ın sırası, okuma sırasına göre yanlış yerde bulunmaktadır. Sıralama yaparken listenin başından itibaren, her elemanın bottom / alt (elemanı kaplayan dörtgenin alanının alt y koordinatı) değeri bir sonraki ile karşılaştırılarak, bottom değeri yüksek olanın üstte kalacağı şekilde sırlama yapıldı. Görselden görülebileceği gibi sıralanmamış listedeki

0. Eleman, sıralanmış listede 1. Eleman olarak yer değiştirdi ve böylece PDF teki gibi mantıksal okuma sırasına göre bileşenler sıralandı.

2.1.3. Öğe listesini satır listesine dönüştürme (get lines)

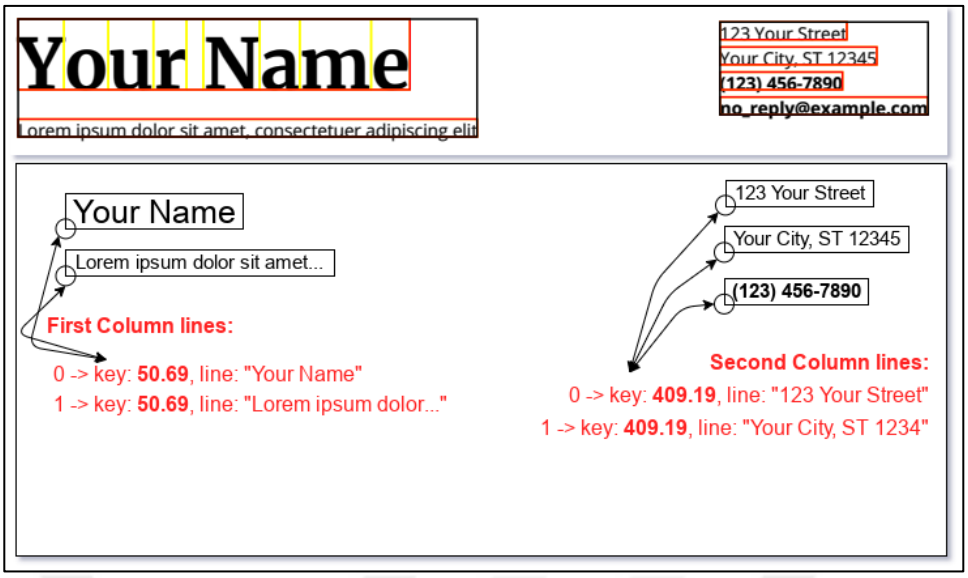
Şekil 2.10'da kırmızı renkle gösterilen satır bileşenlerini oluşturabilmek için sıralamada olduğu gibi items listesinden sırasıyla ikişer eleman alınarak, bu elemanların aynı Y koordinatı uzerinde ve belirli bir X toleransı aralığında olup olmadığı kontrolü yapıldı. Şekil 2.12'de, aynı satırda olan item ların belirlenmesi ve line (satır) yapısının oluşturulması, detaylı bir şekilde gösterildi.



Şekil 2.12. Öğeler aynı satırda mı kontrolünün yapıldığı fonksiyon

2.1.4. Satırları sütun bazında gruplandırmak (split lines)

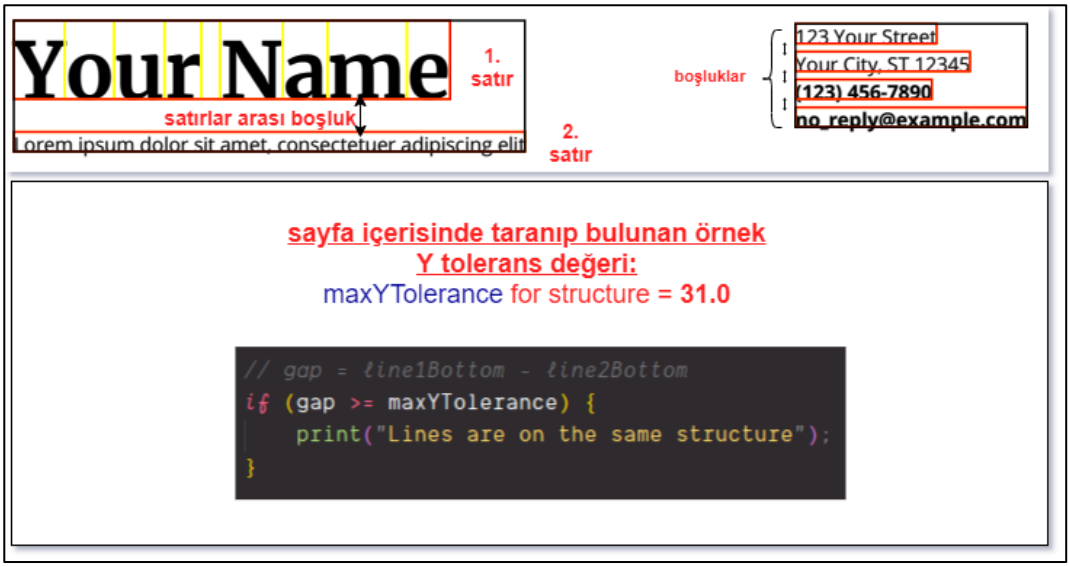
Birden fazla sütunlu PDF dokümanlarında Şekil 2.7'de gösterilen Adım 4'te oluşturulan satır yapılarının X koordinat değerine göre gruplandırılması gerçekleştirildi ve birden fazla sütuna sahip PDF dokümalarındaki sütun sayısından kaynaklanan anlamsız veri oluşma problemi çözüldü. Şekil 2.13'te gruplandırma detaylı bir şekilde gösterildi. Oluşturulan satır yapılarının gruplandırılması, X konum bilgisine göre gerçekleştirildi. Bu adımdan sonra oluşturulacak olan paragraf (structure) yapılarının düzgün oluşturulabilmesi için bu adım önem teşkil etmektedir. Ayni sütun grubunda gruplandırılan satırlar aralarında tekrar gruplandırılarak bir sonraki adımda gösterilen paragraf yapıları oluşturulmaktadır.



Şekil 2.13. Satırların sütunlara göre gruplandırılması

2.1.5. Satırları birleştirerek paragraf oluşturmak (get structures)

Şekil 2.10'da siyah renkle gösterilen paragraf bileşenlerini oluşturabilmek için sıralamada olduğu gibi lines listesinden sırasıyla ikişer eleman alınarak, bu elemanların belirli bir Y toleransı aralığında olup olmadığı kontrolü yapıldı. Şekil 2.14'te, aynı paragrafta (structure da) olan satırların belirlenmesi, detaylı bir şekilde gösterildi.



Şekil 2.14. Paragraf yapılarını oluşturmak

Şekil 2.14'teki paragraf yapıları oluşturulduktan sonra PDF şablon yapısının oluşturulma işlemine geçildi.

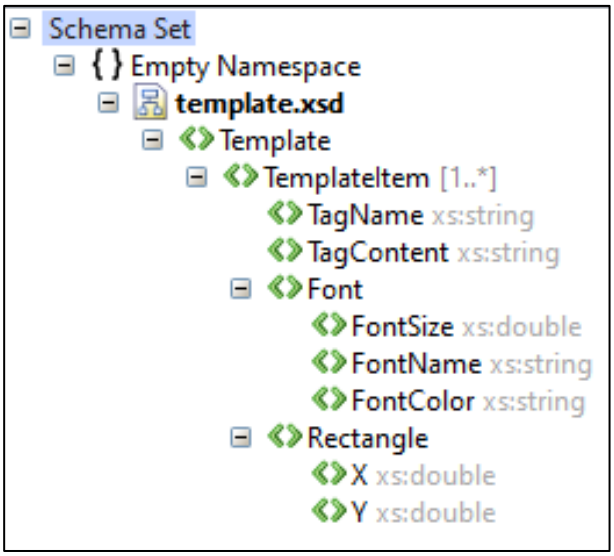
2.1.6. PDF şablon yapısının oluşturulması

PDF şablonunu oluşturmak için kural tabanlı yaklaşım izlendi. Bu yaklaşımda önceden belirlenen kurallara göre, PDF dokümanından okunan verilerin içeriklerinin karşılaştırılması ve yapının oluşturulması sağlandı. PDF şemasının kaydedilip farklı içeriklere uyarlanabilmesi için XML veri yapısı kullanıldı. XML dosyasının kolayca oluşturulabilmesi ve doğrulama işlemlerinin gerçekleştirilebilmesi için XML şeması tanımlandı. Kural tabanlı karşılaştırma yapıldıktan sonra XML yapısında PDF şeması oluşturuldu. Ileri ki aşamalarda kural veritabanının grafik arayüz ve kullanıcı yardımıyla güncellenmesi ve bu sayede daha geniş bir şablon farklılığına sahip PDF dokümanlarının şablon çıkarımının mümkün kılınması amaçlanmaktadır. Şekil 2.15'te önceden tanımlanan XML şeması gösterildi.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Template">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="TemplateItem" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="TagName" type="xs:string"/>
                            <xs:element name="TagContent" type="xs:string"/>
                            <xs:element name="Font">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="FontSize" type="xs:double"/>
                                        <xs:element name="FontName" type="xs:string"/>
                                        <xs:element name="FontColor" type="xs:string"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="Rectangle">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="X" type="xs:double"/>
                                        <xs:element name="Y" type="xs:double"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Şekil 2.15. Tanımlanan PDF şablonu XML şeması (XSD)

Java JAXB kütüphanesi ve IntelliJ IDEA IDE'si kullanılarak Şekil 2.15'teki XSD şeması yardımıyla otomatik Java Pojo class'ları üretildi. Bu classlar yardımıyla da PDF şablonları XML dosyalarına çevrildi.



Şekil 2.16. XSD ağaç görünümü

Şekil 2.16'da XSD şeması ağaç görünümü ile gösterildi. Kök element olarak Template elementi tanımlandı. Template elementinin child elementi olarak ta TemplateItem elementi tanımlandı. TemplateItem elementi şablon içerisindeki her bir Item'ın bilgilerinin tutulduğu elementi teşkil etmektedir. TemplateItem elementine ait format bilgilerinin herbiri ayrı ayrı TagName, TagContent, Font, Rectangle alt elementlerinde tutulmaktadır. Sadece FontSize ve Rectangle elementlerinin double tipinde, diğer elementlerin ise string tipinde tanımlanmış olduğu Şekil 2.16'daki elementlerin sağ kısımlarındaki gri renkle yazılan bilgilerden görülebilmektedir.

```
▼<Template>
  ▶<TemplateItem>...</TemplateItem>
  ▼<TemplateItem>
     <TagName>street</TagName>
     <TagContent>123 Your Street</TagContent>
    ▼<Font>
       <FontSize>10.994999885559082</FontSize>
       <FontName>ProximaNova-Regular</FontName>
       <FontColor>(DeviceRgb) r: 102, g: 102, b: 102, hex: #666666</FontColor>
     </Font>
    ▼<Rectangle>
       <X>72.0</X>
       <Y>654.0</Y>
     </Rectangle>
   </TemplateItem>
  ▶<TemplateItem>...</TemplateItem>
  ▶<TemplateItem>...</TemplateItem>
  ▶<TemplateItem>...</TemplateItem>
  ▶<TemplateItem>...</TemplateItem>
  ▶<TemplateItem>...</TemplateItem>
  ▶<TemplateItem>...</TemplateItem>
  ▶<TemplateItem>...</TemplateItem>
</Template>
```

Şekil 2.17. Oluşturulan PDF XML yapısı

Şekil 2.17’de <Template> listesi içerisinde birden fazla, kural tabanlı karşılaştırma sonucu seçilen <TemplateItem> tagleri gösterildi. Görselde örnek bir <TemplateItem> tagi detaylı bir şekilde sunuldu. Kişiye özel CV dokümanı oluşturulma aşamasında Şekil 2.17’de gösterilen XML yapılı PDF şeması kullanıldı.

Bu XML dosyasındaki <TemplateItem> ve alt tagleri CV dokümanında belirli metinlerin, PDF dokümanının neresinde, hangi yazı tipi, yazı boyutu ve yazı rengi formatında oluşturulması gerektiği bilgisini içermektedir.

<TagName> tagi PDF dokümanında belirli metinlerin ayrımını yapabilmek için ayarlanan etiket ismi ya da etiket id si olarak tanımlandı. Örneğin kişisel CV dokümanı oluştururken ‘telefon numarası’ metnine ait bilgileri XML dosyasından çekebilmek için XML dosyasında TagName’i ‘phone’ olan TemplateItem’ı bulmak ve yazı tipi, lokasyon bilgileri gibi bilgilere ulaşıp ordaki bilgileri kullanarak telefon numarasının PDF dokümanında oluşturulması yolu izlendi.

<TagContent> elementi, orjinal CV’ye ait PDF şablonu çıkartma aşamasında oluşturulan herhangi bir yapının metin içeriğini temsil etmektedir. Örneğin orjinal CV dokümanından şablon yapısı çıkartılırken ki ‘telefon numarası’ kısmındaki metnin içeriği olarak özetlenebilir.

<Font> elementi, yine belirli bir metne (yapıya) ait yazı tipi bilgilerini içinde bulundurmaktadır. Aşağıda Font elementi içerisinde tanımlanan diğer elementler ve bigileri sunuldu.

- <FontSize> elementi, yazı boyutu bilgisini
- <FontName> elementi, yazı tipi bilgisini
- <FontColor> elementi, yazı rengi bilgisini içermektedir.

<Rectangle> elementi, isminin verilme sebebi kullanılan java kütüphanesi PDF dokümanından okunan verilerin lokasyon bilgisini Rectangle isimli nesne yapısında sunmasından kaynaklanmaktadır. Bu element, yine belirli bir metnin (yapının) PDF dokümanının xy düzlemi içerisinde, tam olarak hangi koordinatlarda bulunduğu bilgisini içermektedir. Aşağıda Font elementi içerisinde tanımlanan diğer elementler ve bigileri sunuldu.

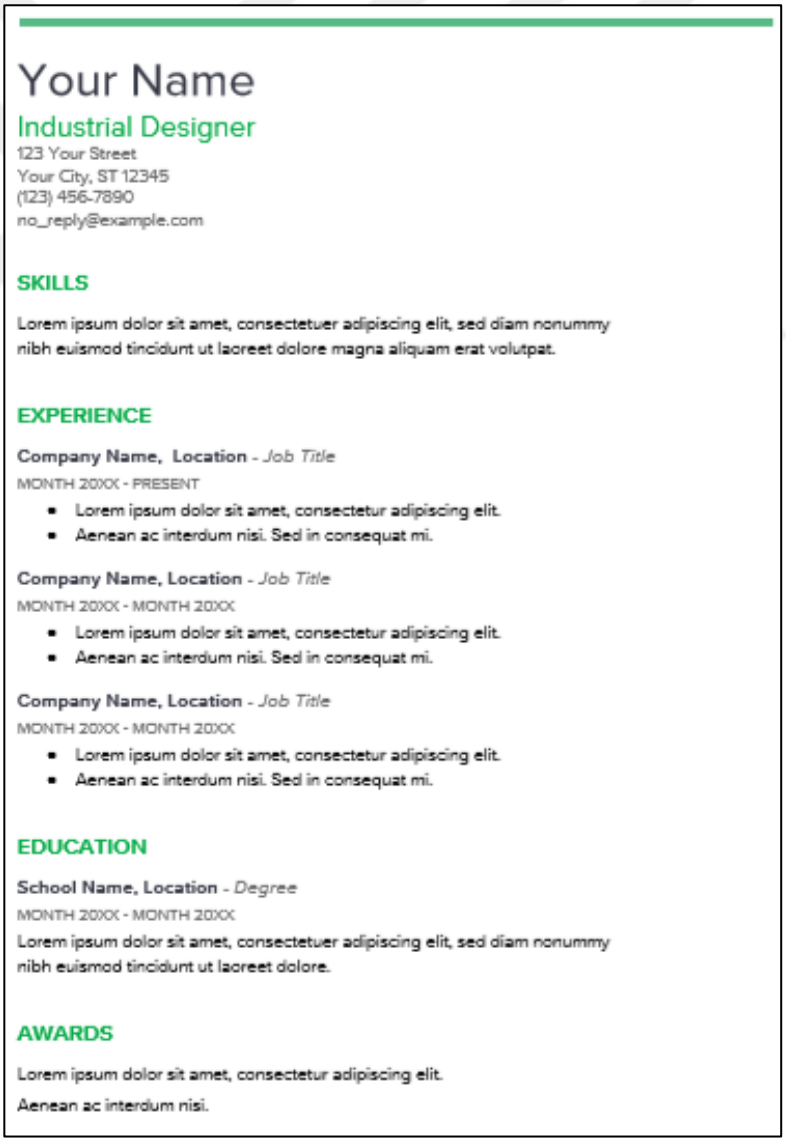
- <X> elementi, bileşenin xy düzlemindeki X koordinatını
- <Y> elementi, bileşenin xy düzlemindeki Y koordinatını teşkil etmektedir.

2.2. Örnek CV Dokümanları Üzerinden Şablon Çıkarımı

2.2.1 ve 2.2.2 başlıkları altında tek ve iki sütunlu CV örnekleri üzerinden şablon çıkarımı adımları görsellerle desteklenerek sunulmuştur.

2.2.1. Tek sütunlu CV örneği

İlk örnek olarak tek sütunlu yapıya sahip bir CV örneği seçildi. Şekil 2.18'de örnek CV'nin içeriği gösterilmektedir.



Your Name

Industrial Designer

123 Your Street
Your City, ST 12345
(123) 456-7890
no_reply@example.com

SKILLS

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

EXPERIENCE

Company Name, Location - *Job Title*
MONTH 20XX - PRESENT
- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Aenean ac interdum nisi. Sed in consequat mi.

Company Name, Location - *Job Title*
MONTH 20XX - MONTH 20XX
- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Aenean ac interdum nisi. Sed in consequat mi.

Company Name, Location - *Job Title*
MONTH 20XX - MONTH 20XX
- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Aenean ac interdum nisi. Sed in consequat mi.

EDUCATION

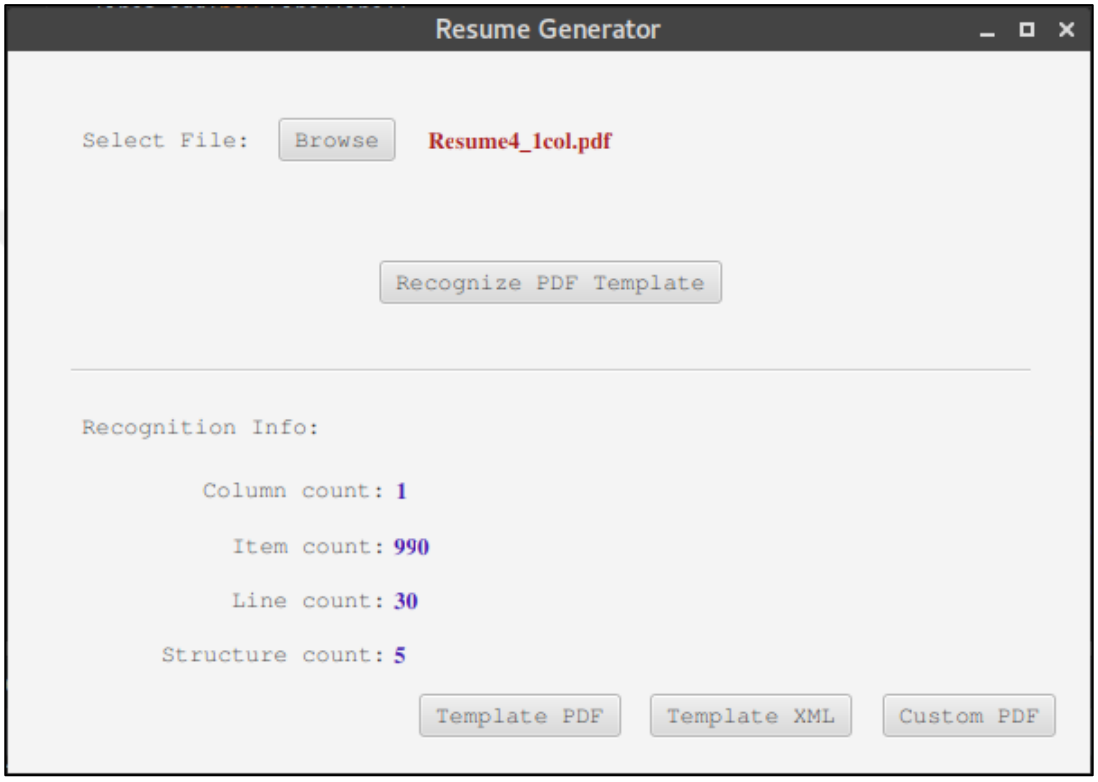
School Name, Location - *Degree*
MONTH 20XX - MONTH 20XX
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore.

AWARDS

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Aenean ac interdum nisi.

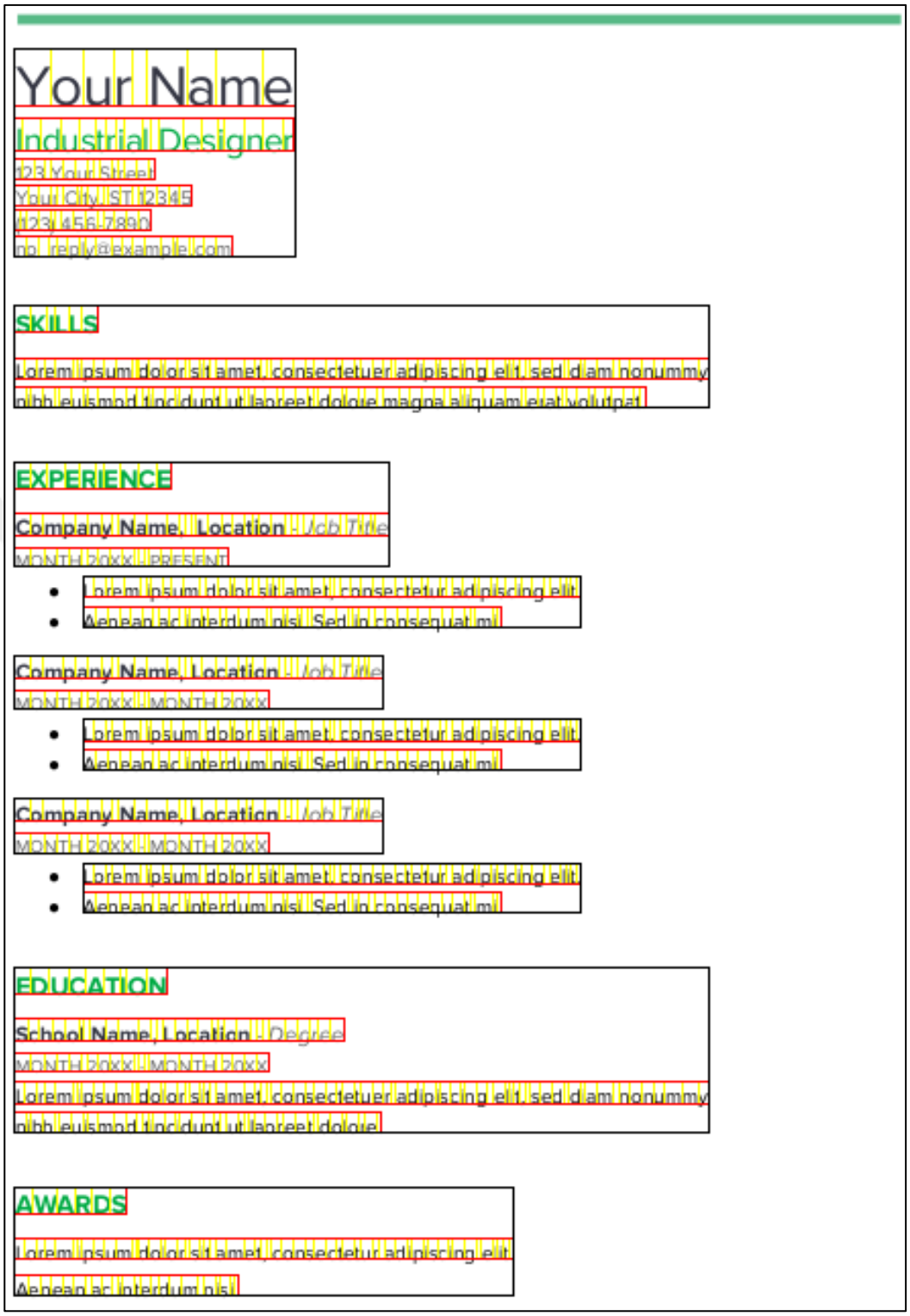
Şekil 2.18. CV Örnek 1 – PDF içeriği

İlk adım olarak uygulama arayüzünden, şablon çıkarımı gerçekleştirilecek olan Şekil 2.18'de gösterilen PDF formatlı CV örneği girdi olarak seçilmektedir. Girdi seçildikten sonra "Recognize PDF Template" buttonu yardımıyla tek tıkla Şekil 2.7'de gösterilen tüm algoritma adımları uygulanmaktadır ve uygulama ana erkanında Şekil 2.19'da gösterildiği gibi algoritma sonucu elde edilen Column, Item, Line ve Structure bilgileri gösterilmektedir.



Şekil 2.19. CV Örnek 1 – Uygulama ana ekranı çıktısı

Şekil 2.19'da "Recognition Info" sekmesindeki bilgilerin Şekil 2.18'deki CV örneğiyle eşleştiği görülebilmektedir. Aynı CV örneğinin docx formatı Microsoft Office Word uygulaması ile açıldı ve uygulamanın Status Bar'ında gösterilen satır sayısı Şekil 2.19'daki Line count bilgisiyle eşleştiği gözlemlendi. Uygulama ana erkanının sağ alt bölümünde üç farklı çıktı gösteren 3 farklı button bulunmaktadır. "Template PDF" butonu yardımıyla şablon çıkarım işleminin görselleştirilmiş PDF hali, "Template XML" butonu yardımıyla çıkarımı yapılan şablonun XML formatına dönüştürülmüş hali ve "Custom PDF" butonu yardımıyla da çıkarımı yapılan şablon bilgileri kullanılarak kişiye özel oluşturulan PDF örneği görüntülenebilir.



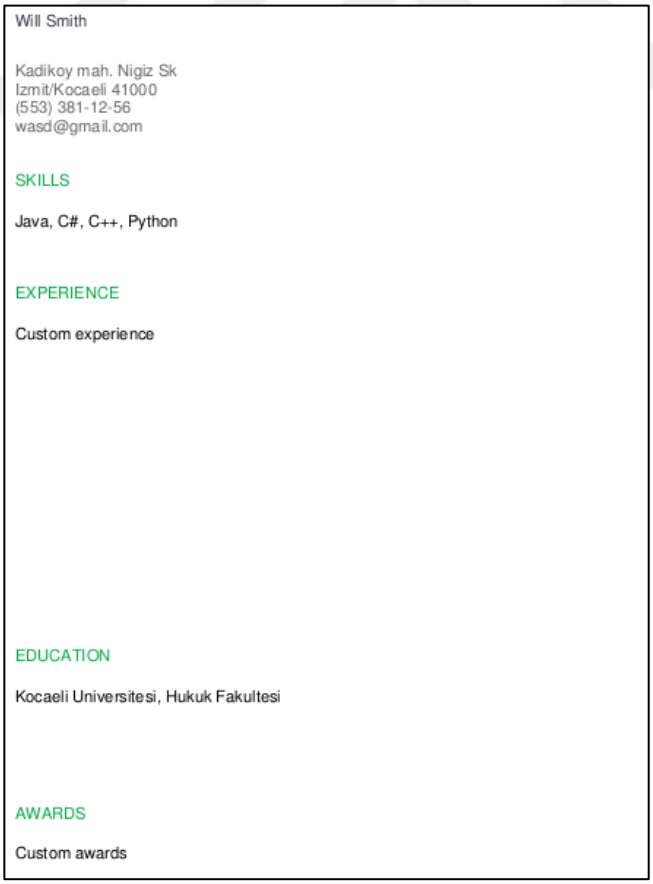
Şekil 2.20. CV Örnek 1 – Şablon çıkarımının görselleştirilmiş hali

Şekil 2.20'de, yazdığımız algoritma yardımıyla Şekil 2.18'deki PDF örneğinin şablon çıkarımının dikdörtgen yapıları kullanılarak görselleştirilmiş hali sunuldu. Paragraf yapıları siyah, satırlar kırmızı ve harfler sarı renkte gösterildi.

```
<Template>
  <TemplateItem>
    <TagName>name</TagName>
    <TagContent>Your Name</TagContent>
    <Font>
      <FontSize>30.0</FontSize>
      <FontName>ProximaNova-Regular</FontName>
      <FontColor>(DeviceRgb) r: 53, g: 55, b: 68, hex: #353744</FontColor>
    </Font>
    <Rectangle>
      <X>72.0</X>
      <Y>693.75</Y>
    </Rectangle>
  </TemplateItem>
  <TemplateItem>...</TemplateItem>
  <TemplateItem>...</TemplateItem>
  <TemplateItem>...</TemplateItem>
  <TemplateItem>...</TemplateItem>
  <TemplateItem>...</TemplateItem>
  <TemplateItem>...</TemplateItem>
  <TemplateItem>...</TemplateItem>
</Template>
```


Şekil 2.21. CV Örnek 1 – Şablonun XML'e dönüştürülmüş hali

Şekil 2.9'daki bilgiler göz önünde bulundurularak, Şekil 2.21'deki "Your Name" metnine ait TagName, Font ve Rectangle(konum) XML bilgileri, Şekil 2.18'deki örnek CV dokümanı ile karşılaştırıldığında bilgilerin eşleştiği görülebilmektedir.

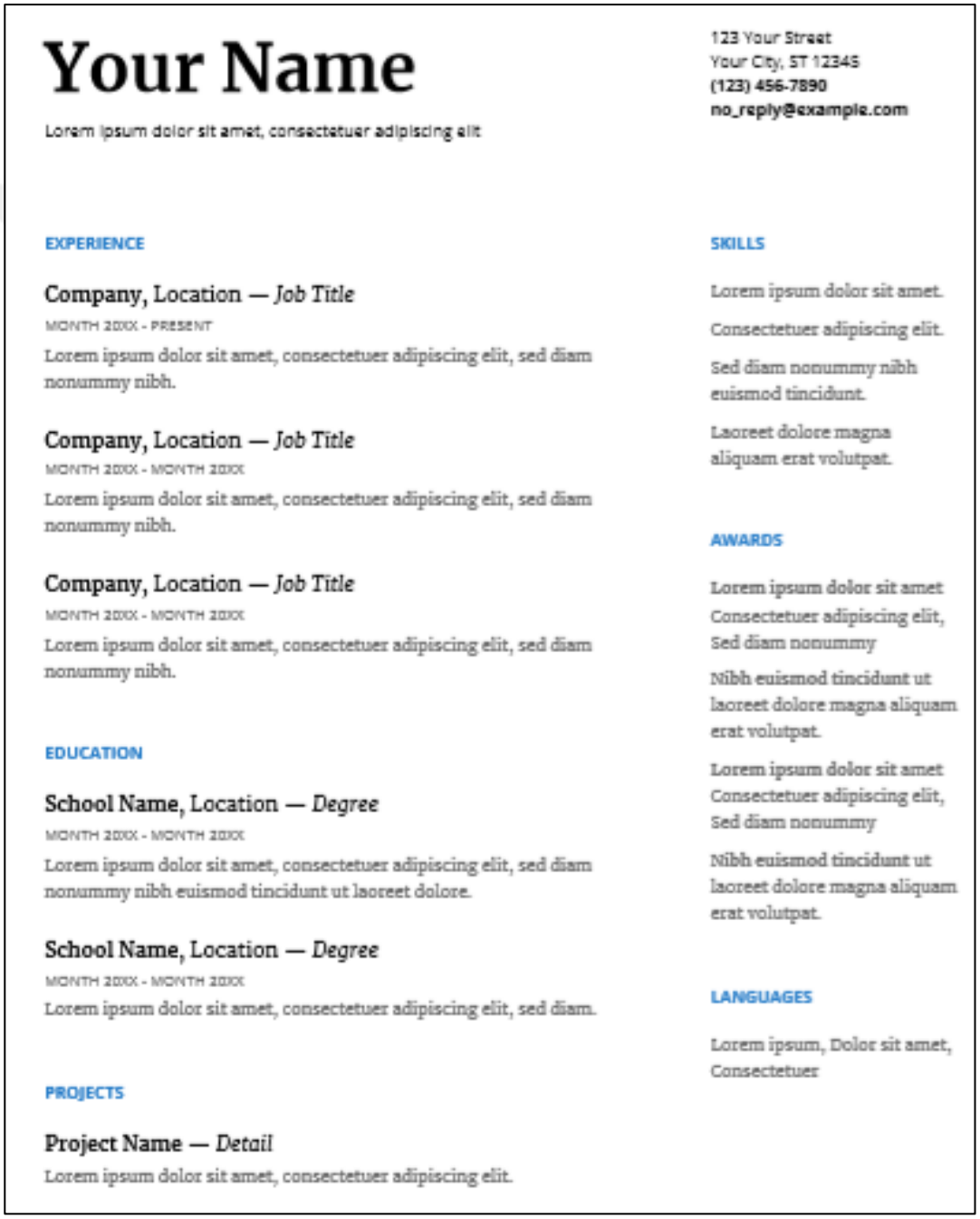


Şekil 2.22. CV Örnek 1 – Kişiye özel oluşturulmuş PDF

Şekil 2.22'de kişiye özel CV bilgileri üzerine Şekil 2.21'deki şablon bilgileri giydirilerek oluşturulan kişiye özel PDF dokümanı örneği gösterilmektedir.

2.2.2. İki sütunlu CV örneği

İkinci örnek olarak ta 2 sütunlu yapıya sahip bir CV örneği seçildi. Şekil 2.23'te örnek CV'nin içeriği gösterilmektedir.



Your Name

Lorem ipsum dolor sit amet, consectetuer adipiscing elit

123 Your Street
Your City, ST 12345
(123) 456-7890
no_reply@example.com

EXPERIENCE

Company, Location — *Job Title*
MONTH 20XX - PRESENT
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh.

Company, Location — *Job Title*
MONTH 20XX - MONTH 20XX
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh.

Company, Location — *Job Title*
MONTH 20XX - MONTH 20XX
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh.

EDUCATION

School Name, Location — *Degree*
MONTH 20XX - MONTH 20XX
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore.

School Name, Location — *Degree*
MONTH 20XX - MONTH 20XX
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam.

PROJECTS

Project Name — *Detail*
Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

SKILLS

Lorem ipsum dolor sit amet.

Consectetuer adipiscing elit.

Sed diam nonummy nibh euismod tincidunt.

Laoreet dolore magna aliquam erat volutpat.

AWARDS

Lorem ipsum dolor sit amet
Consectetuer adipiscing elit, Sed diam nonummy

Nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Lorem ipsum dolor sit amet
Consectetuer adipiscing elit, Sed diam nonummy

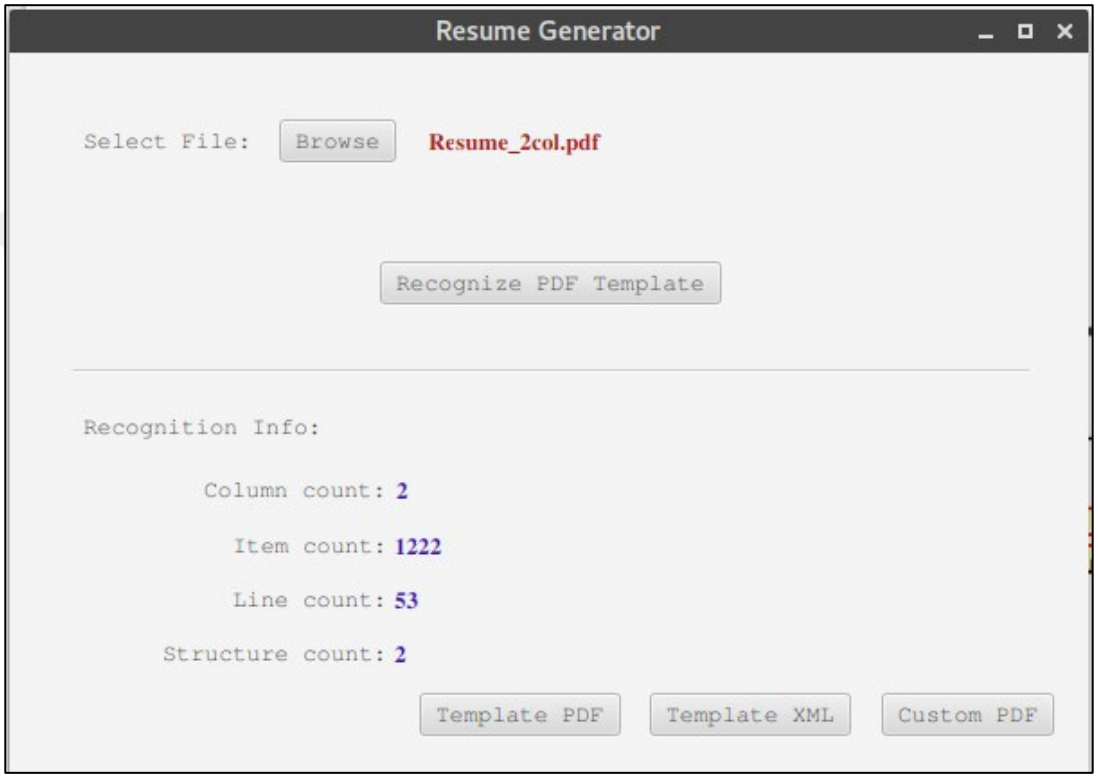
Nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

LANGUAGES

Lorem ipsum, Dolor sit amet, Consectetuer

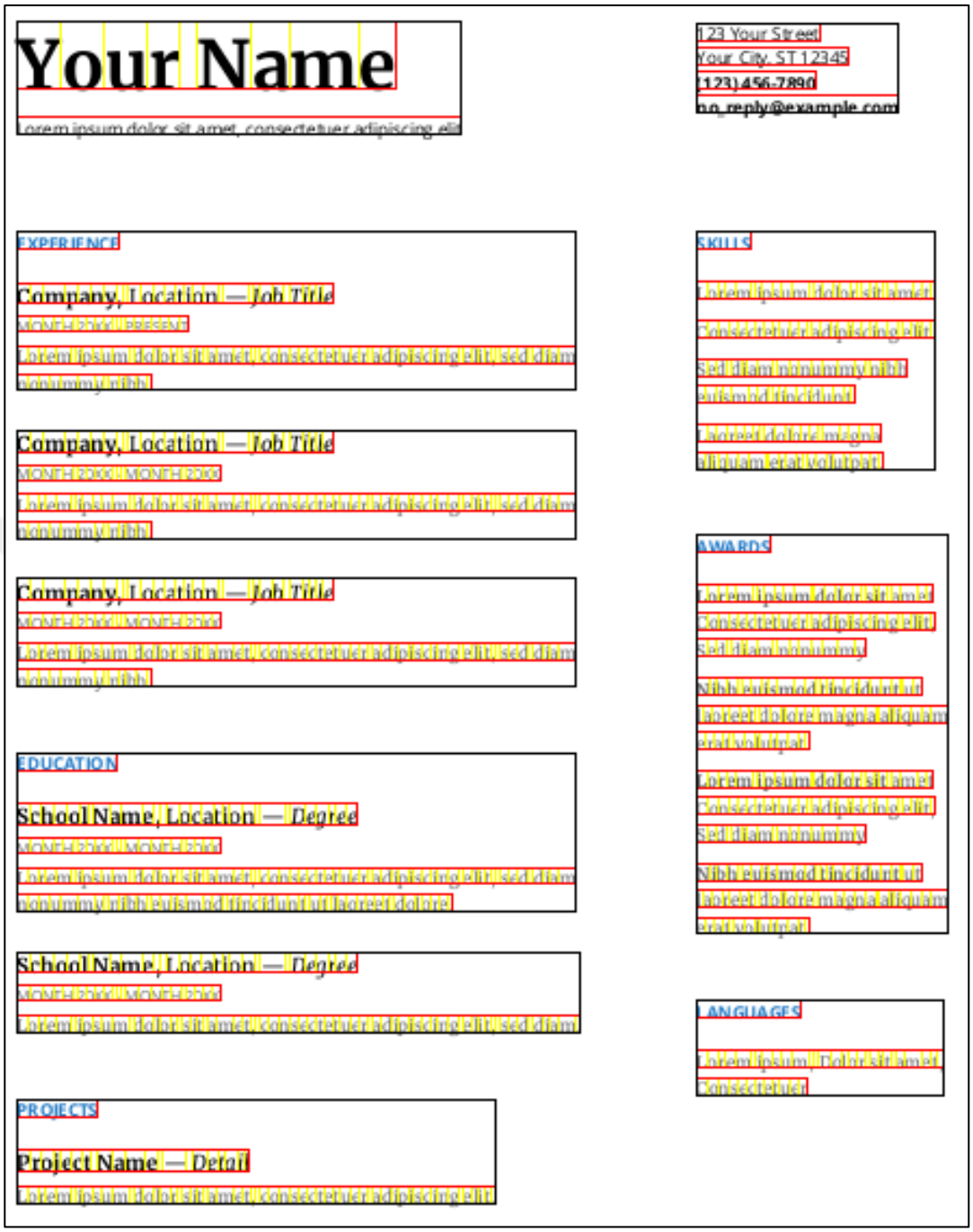
Şekil 2.23. CV Örnek 2 – PDF içeriği

İlk adım olarak uygulama arayüzünden, şablon çıkarımı gerçekleştirilecek olan PDF formatlı CV örneği girdi olarak seçilmektedir. Girdi seçildikten sonra “Recognize PDF Template” buttonu yardımıyla tek tıkla Şekil 2.7’de gösterilen tüm algoritma adımları uygulanmaktadır ve uygulama ana erkanında Şekil 2.24’te gösterildiği gibi algoritma sonucu elde edilen Column, Item, Line ve Structure bilgileri gösterilmektedir.



Şekil 2.24. CV Örnek 2 – Uygulama ana ekranı çıktısı

Şekil 2.24’te “Recognition Info” sekmesindeki bilgilerin Şekil 2.23’teki CV örneğiyle eşleştiği görülebilmektedir. Uygulama ana erkanının sağ alt bölümünde üç farklı çıktı gösteren 3 farklı button bulunmaktadır. “Template PDF” butonu yardımıyla şablon çıkarım işleminin görselleştirilmiş PDF hali, “Template XML” butonu yardımıyla çıkarımı yapılan şablonun XML formatına dönüştürülmüş hali ve “Custom PDF” butonu yardımıyla da çıkarımı yapılan şablon bilgileri kullanılarak kişiye özel oluşturulan PDF örneği görüntülenebilir. Şekil 2.24’te ‘Recognition Info’ bölümü içerisinde bulunan ‘Column count: 2’ bilgisinden de görülebileceği gibi bu örnekte şablon taraması yapılan Şekil 2.23’teki PDF dokümanının 2 sütunlu olduğu bilgisi doğru tespit edilmiştir.

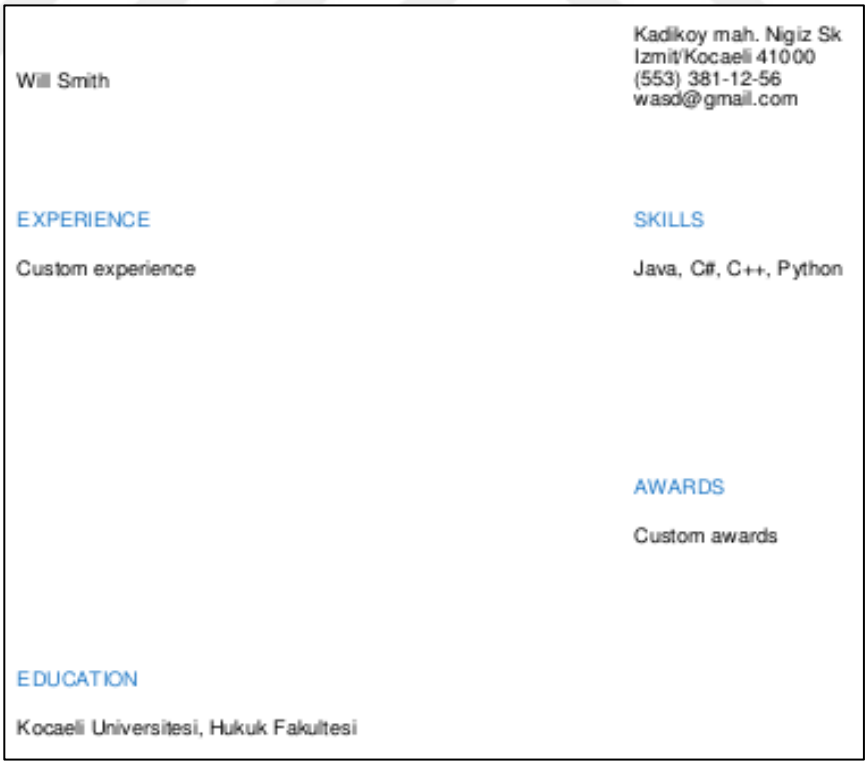


Şekil 2.25. CV Örnek 2 – Şablon çıkarımının görselleştirilmiş hali

Şekil 2.25'te, yazdığımız algoritma yardımıyla Şekil 2.23'teki PDF örneğinin şablon çıkarımının dikdörtgen yapıları kullanılarak görselleştirilmiş hali sunuldu. Paragraf yapıları siyah, satırlar kırmızı ve harfler sarı renkte gösterildi. Örnekteki PDF 2 sütunlu yapıya sahiptir ki bu örnekle iki sütunlu PDF girdileri için de algoritmanın çalıştığı gözlemlenebilir. Şablondaki bilgileri sütun bazlı gruplandırabilmek için de Şekil C.10'de gösterilen splitColumnLinesByX fonksiyonu kullanıldı.

Şekil 2.26. CV Örnek 2 – Şablonun XML'e dönüştürülmüş hali

Şekil 2.9'daki bilgiler göz önünde bulundurularak, Şekil 2.26'daki "EXPERIENCE" metnine ait TagName, Font ve Rectangle(konum) XML bilgileri, Şekil 2.23'teki örnek CV dokümanı ile karşılaştırıldığında bilgilerin eşleştiği görülebilmektedir.



Şekil 2.27. CV Örnek 2 – Kişiye özel oluşturulmuş PDF

Şekil 2.27'de kişiye özel CV bilgileri üzerine Şekil 2.26'daki şablon bilgileri giydirilerek oluşturulan kişiye özel PDF dokümanı örneği gösterilmektedir.

3. BULGULAR VE TARTIŞMA

Yapılan çalışmada PDF içeriği kütüphane yardımıyla, direk doğal PDF yapısı içerisinden okunduğu için, PDF belgelerini, resim veya HTML`e dönüştürüp sonra OCR vs. gibi tekniklerle işlemek yerine, içeriği doğrudan PDF`ten çıkarmak ve analiz etmenin daha kolay ve kesin yöntem olduğu sonucu doğrulandı. PDF içerisinden okunup tag lenen öğeler önceden tanımlı kurallar doğrultusunda gerçekleştirildi ki kural veri tabanının grafik bir arayüz yardımıyla kullanıcı tarafından güncellenmesi ile daha geniş çapta PDF şablon türlerinin tanınmasına olanak sağlayabileceği sonucuna varıldı. Ayrıca okunan öğelerin yazı tipi stili meta datası da PDF dokümanından okunabildiği için daha tutarlı bir şekilde kişiye özel yeni CV dokümanlarının oluşturulması gerçekleştirildi. Kullanılan Google Font API (Application Programming Interface, Yazılım Programlama Arayüzü)'si yardımıyla orjinal yazı tipi gerçek zamanlı indirilip, yazı tipinde bile bir farklılık oluşturmamaya özen gösterildi.

4. SONUÇLAR VE ÖNERİLER

Ileriki çalışmalarda kural tabanlı yaklaşımın, literatürdeki farklı yaklaşımlar ile birlikte kullanılması, görüntü işleme ve NLP (Doğal dil işleme) teknolojilerinin çalışmaya dahil edilmesi ile çok daha tutarlı şablonların elde edilmesi öngörülmektedir. Yapılan çalışmada önceden belirlenmiş kural tabanlı içerik eşleştirilmesi gerçekleştirildi. Kural tabanlı eşleştirme yerine Makine öğrenmesi gibi yöntemler kullanılıp bu işlemin otomatikleştirilmesi hem projenin tamamen otomatize edilmesi açısından hem de daha geniş çapta PDF dokümanlarının tanınması açısından önem arz etmektedir. Raster yapılı PDF dokümanlardan içerik veya şablon çıkarma gibi konuların ele alınabilmesi için OCR ve benzeri tekniklerin kullanılması ve böylece daha geniş çapta PDF doküman tanıma veya içerik çıkarmanın sağlanması gerçekleştirilebilir. Yazı tiplerinin çeşitliliğinden dolayı farklı font indirme API'leri dahil edilerek, daha geniş çapta yazı tiplerinin desteklenmesi de gerçekleştirilebilir.

KAYNAKLAR

[1] Tunçer M., "Özgeçmiş Hazırlama Tüyoları ve CV Örneği, https://www.kariyer.net/kariyer-rehberi/ozgecmis-hazirlama-tuyolari-ve-cv-ornegi/, (Ziyaret tarihi: 18 Aralık 2019).

[2] Mohemad R., Hamdan A. R., Othman Z. A., Mohamad N. M., Automatic Document Structure Analysis of Structured PDF Files, *IJNCAA,* 2011, **1**(2) 404-411.

[3] Hassan T., Object-Level Document Analysis of PDF Files, *9th ACM symposium on Document engineering,* Munich, Germany, 2009.

[4] Liu Y., Bai K., Mitra P., Giles C. L., Improving the Table Boundary Detection in PDFs by Fixing the Sequence Error of the Sparse Lines, *10th International Conference on Document Analysis and Recognition*, Barcelona, Spain, 26-29 Temmuz 2009.

[5] Jiang D., Yang X., Converting PDF to HTML approach based on text detection, *In Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human (ICIS '09)*, Seoul, Korea, 24-26 Kasım 2009.

[6] Chao H., Fan J., Layout and Content Extraction for PDF Documents *Document Analysis Systems VI 6th International Workshop*, Florence, Italy, 8-10 Eylül 2004.

[7] Aiello M., Monz C., Todoran L., Worring M., Document Understanding for a Broad Class of Documents, *International Journal on Document Analysis and Recognition,* 2002, **5**(1), 1-16.

[8] Altamura O., Esposito F., Malerba D., Transforming paper documents into XML format with WISDOM++, *International Journal on Document Analysis and Recognition,* 2000, **4**(1), 2-17.

[9] Gabdulkhakova A., Tamir H., Document understanding of graphical content in natively digital PDF documents, *2012 ACM symposium on Document engineering*, Paris, France, Eylül 2012.

[10] Baker J. B., Sexton A. P., Sorge V., M. Suzuki, Comparing Approaches to Mathematical Document Analysis from PDF, *2011 International Conference on Document Analysis and Recognition*, Beijing, China, 2011.

[11] Constantin A., Pettifer S., A. Voronkov, PDFX: fully-automated PDF-to-XML conversion of scientific literature, *2013 ACM symposium on Document engineering*, Florence, Italy, Eylül 2013.

[12] https://en.wikipedia.org/wiki/PDF, (Ziyaret tarihi: 26 Aralık 2019).

[13] http://portal.netcad.com.tr/pages/viewpage.action?pageId=106727005, (Ziyaret tarihi: 2 Kasım 2020).

[14] https://visual-integrity.com/spotting-difference-vector-raster-pdf/, (Ziyaret tarihi: 11 Şubat 2020).

[15] http://www.printmyfolders.com/understanding-pdf, (Ziyaret tarihi: 26 Aralık 2019).

[16] Anand A., https://securityxploded.com/pdf_internals.php, (Ziyaret tarihi: 26 Aralık 2019).

[17] https://lotabout.me/orgwiki/pdf.html, (Ziyaret tarihi: 27 Aralık 2019).

[18] https://tika.apache.org/1.7/formats.html, (Ziyaret tarihi: 11 Şubat 2020).

[19] https://www.glyphandcog.com/textext.html, (Ziyaret tarihi: 11 Şubat 2020).

[20] https://stackoverflow.com/questions/22340674/performance-itext-vs-pdfbox, (Ziyaret tarihi: 11 Şubat 2020).

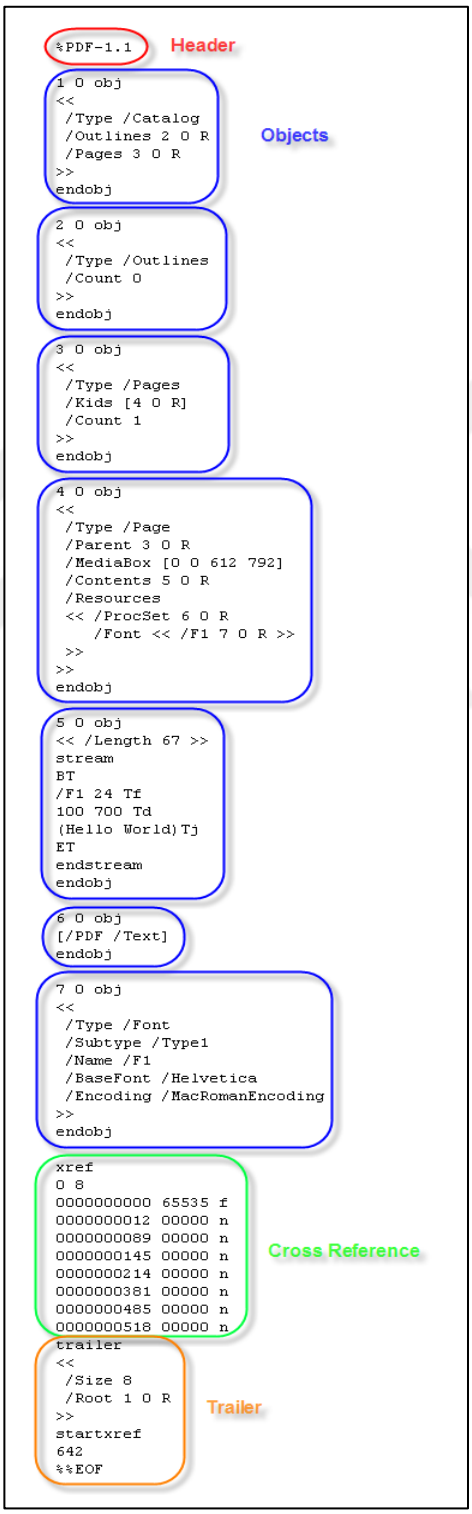
[21] Stevens D., https://blog.didierstevens.com/2008/04/09/quickpost-about-the-physical-and-logical-structure-of-pdf-files/, (Ziyaret tarihi: 27 Aralık 2019).

[22] Lukan D., https://resources.infosecinstitute.com/pdf-file-format-basic-structure/, (Ziyaret tarihi: 27 Aralık 2019).

EKLER

Ek-A

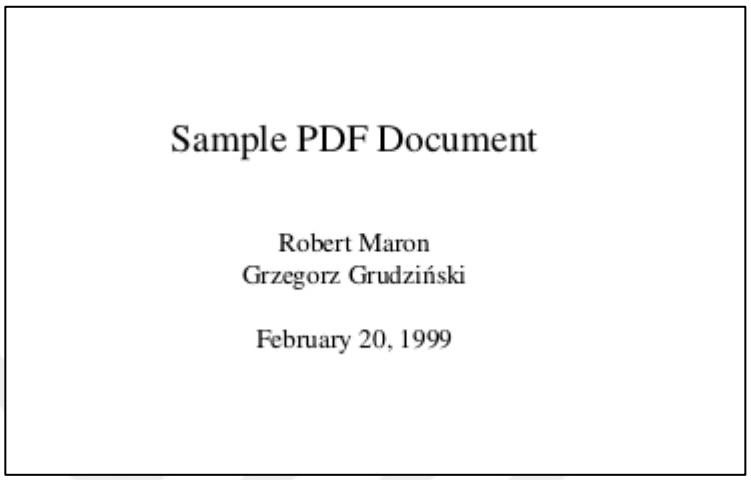
Şekil A.1'de içeriği bir metin editörü ile görüntülenen basit bir "Merhaba Dünya" PDF örneği gösterilmektedir.



Şekil A.1. PDF içeriği (txt) [21]

Ek-B

Şekil B.1'de örnek bir PDF belgesi ve belgeye ait cross-reference ile trailer kısımları gösterilmektedir:



Şekil B.1. Örnek PDF belgesi [22]

Cross-reference ve trailer bölümleri Şekil B.2'de sunulmaktadır:

```
1405 xref
1406 0 223
1407 0000000000 65535 f
1408 0000001741 00000 n
1409 ...
1410 0000049957 00000 n
1411 0000050089 00000 n
1412 trailer
1413 <<
1414 /Size 223
1415 /Root 221 0 R
1416 /Info 222 0 R
1417 >>
1418 startxref
1419 50291
1420 %%EOF
```

Şekil B.2. Cross-reference ve trailer içeriği [22]

Ek-C

Şekil C.1'de PDF şablon yapısını çıkarma algoritmasının ilk aşaması olan TextItem nesnesini oluşturma fonksiyonu gösterilmektedir.

```
public TextItem(TextRenderInfo textRenderInfo, float pageHeight) {
    this.pageHeight = pageHeight;
    this.textRenderInfo = textRenderInfo;
    baseline = textRenderInfo.getBaseline().getStartPoint().get(1);
    realRectangle = getRectangle(textRenderInfo);
    drawableRectangle = getDrawableRectangle();
    text = textRenderInfo.getText();
    fontSize = getFontSize(textRenderInfo);
    font = getFont(textRenderInfo);
}
```

Şekil C.1. TextItem fonksiyonu

Şekil C.2'de TextItem nesnesinin PDF dokümanındaki konum bilgisinin tutulduğu realRectangle değişkeninin oluşturulma algoritması gösterilmektedir.

```
static Rectangle getRectangle(TextRenderInfo textRenderInfo) {
    LineSegment descentLine = textRenderInfo.getDescentLine();
    LineSegment ascentLine = textRenderInfo.getAscentLine();
    float x0 = descentLine.getStartPoint().get(0);
    float x1 = descentLine.getEndPoint().get(0);
    float y0 = textRenderInfo.getBaseline().getStartPoint().get(Vector.I2);
    float y1 = ascentLine.getEndPoint().get(1);
    return new Rectangle(x0, y0, x1 - x0, y1);
}
```

Şekil C.2. getRectangle fonksiyonu

TextItem nesnesinin konumu PDF dokümanından okunduğu zaman bu konum bilgisi tam olarak PDF dosyasını görüntüleyen kişinin gördüğü konum bilgisi olmayabiliyor. Yani PDF dokümanından okunan konum bilgisi ve PDF üzerinde gördüğümüz TextItem konum bilgisi aynı değerlere sahip olmayabiliyor. Şekil C.3'te TextItem konum bilgisini elde edebilmek için uygulanan algoritma gösterilmektedir:

```
public Rectangle getDrawableRectangle() {
    if (realRectangle != null) {
        float x0 = getRealRectangle().getLeft();
        float y0 = pageHeight - getRealRectangle().getBottom() - getFontSize();
        float width = getRealRectangle().getRight() - getRealRectangle().getLeft();

        Rectangle drawableRect = new Rectangle(x0, y0, width, getFontSize());
        return drawableRect;
    }
    return null;
}
```

Şekil C.3. getDrawableRectangle fonksiyonu

Şekil C.4'te TextItem'a ait yazı tipi boyutunun elde edilme fonksiyonu gösterilmektedir:

```
static float getFontSize(TextRenderInfo textRenderInfo) {
    Matrix textToUserSpaceTransformMatrix = textRenderInfo.getGraphicsState().getCtm();
    float fontSize = new Vector(0, textRenderInfo.getGraphicsState().getFontSize(), 0)
            .cross(textToUserSpaceTransformMatrix)
            .length();
    return fontSize;
}
```

Şekil C.4. getFontSize fonksiyonu

Şekil C.5'te TextItem'ları mantıksal okuma sırasına göre dizmeyi sağlayan item lar arası karşılaştırma fonksiyonu gösterilmektedir:

```
@Override
public int compareTo(MyItem o) {
    double left = getLL().getX();
    double bottom = getLL().getY();
    double oLeft = o.getLL().getX();
    double oBottom = o.getLL().getY();

    if (bottom - oBottom > itemPositionTolerance)
        return -1;
    else if (oBottom - bottom > itemPositionTolerance)
        return 1;
    else
        return Double.compare(left, oLeft);
}
```

Şekil C.5. compareTo fonksiyonu

Şekil C.6'da TextItem elementlerini birleştirip Line (satır) yapıları oluşturmaya yarayan algoritma gösterilmektedir:

```
public List<Line> getLines(List<MyItem> items) {
    List<Line> lines = new ArrayList<>();
    List<MyItem> line = new ArrayList<>();
    for (MyItem item : items) {
        if (line.isEmpty()) {
            line.add(item);
            continue;
        }
        if (areOnSameLine(line.get(line.size() - 1), item)) {
            line.add(item);
        } else {
            lines.add(new Line(line));
            line = new ArrayList<>();
            line.add(item);
        }
    }
    if (!line.isEmpty())
        lines.add(new Line(line));
    return lines;
}
```

Şekil C.6. getLines fonksiyonu

Şekil C.7'de birden fazla sütun yapısına sahip PDF dokümanlarında Line elementlerinin sütun bazında gruplandırılma algoritması gösterilmektedir.

```
public static HashMap<Double, List<Line>> splitColumnLinesByX(List<Line> lines) {
    HashMap<Double, List<Line>> multipleColumnLines = new HashMap<>();
    for (Line line : lines) {
        double lineX = Math.round(line.getLL().getX() * 100.0) / 100.0;

        if (!multipleColumnLines.containsKey(lineX)) {
            List<Line> oneColumnLines = new ArrayList<>();
            multipleColumnLines.put(lineX, oneColumnLines);
        }
        multipleColumnLines.get(lineX).add(line);
    }

    return multipleColumnLines;
}
```

Şekil C.7. splitColumnLinesByX fonksiyonu

Şekil C.8'de Line elementlerini birleştirip Structure (paragraf) yapıları oluşturmaya yarayan algoritma gösterilmektedir:

```
public List<Structure> getStructures(List<Line> lines, Double maxYTolerance) {
    List<Structure> structures = new ArrayList<>();
    List<MyItem> structure = new ArrayList<>();
    for (Line line : lines) {
        if (structure.isEmpty()) {
            structure.add(line);
            continue;
        }
        if (areInSameStructure((Line) structure.get(structure.size() - 1), line, maxYTolerance)) {
            structure.add(line);
        } else {
            structures.add(new Structure(structure));
            structure = new ArrayList<>();
            structure.add(line);
        }
    }
    if (!structure.isEmpty())
        structures.add(new Structure(structure));
    return structures;
}
```

Şekil C.8. getStructures fonksiyonu

Ek-D

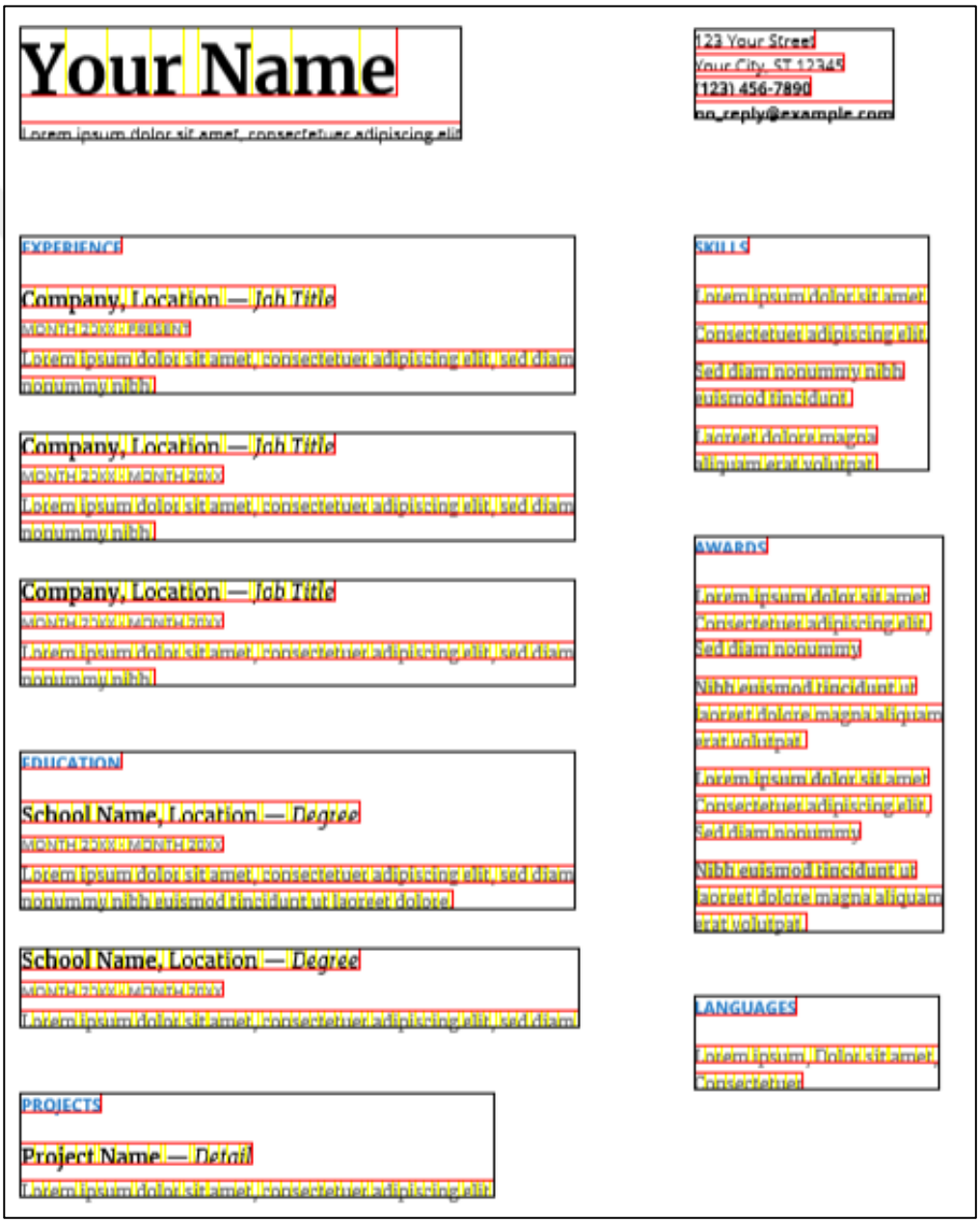
Şekil D.1'de önceden tasarlanmış XSD dosyası kullanılarak otomatik bir şekilde elde edilen CV şablon yapısının XML yapısına dönüştürülme algoritması gösterilmektedir:

```
public static void createXmlTest(HashMap<String, Structure> multipleColumnStructures) {
    DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder docBuilder = null;
        docBuilder = docFactory.newDocumentBuilder();
        Document doc = docBuilder.newDocument();

        Template template = new Template();
        for (Map.Entry<String, Structure> structureEntry : multipleColumnStructures.entrySet()) {
            Structure structure = structureEntry.getValue();
            for (MyItem subItem : structure.getItems()) {
                for (TagContent tagContentItem : TagContent.getTagContentList()) {
                    for (String tagContent : tagContentItem.getContent()) {
                        if (tagContent.toLowerCase().contains(subItem.getText().toLowerCase())) {
                            Template.TemplateItem templateItem = new Template.TemplateItem();
                            templateItem.setTagName(tagContentItem.getTagName());
                            templateItem.setTagContent(subItem.getText());

                            Template.TemplateItem.Font font = new Template.TemplateItem.Font();
                            font.setFontColor(PdfFontService.getFontColor(subItem.getTextRenderInfo()));
                            font.setFontName(subItem.getFont().getFontProgram().getFontNames().getFontName());
                            font.setFontSize(subItem.getFontSize());
                            templateItem.setFont(font);

                            Template.TemplateItem.Rectangle rectangle = new Template.TemplateItem.Rectangle();
                            rectangle.setX(subItem.getLL().getX());
                            rectangle.setY(subItem.getLL().getY());
                            templateItem.setRectangle(rectangle);
                            // add templateItem to templateItems array
                            template.getTemplateItem().add(templateItem);
                        }
                    }
                }
            }
        }
        // Transform Document to XML String
        TransformerFactory tf = TransformerFactory.newInstance();
        Transformer transformer = tf.newTransformer();
        DOMSource domSource = new DOMSource(doc);
        File xmlDir = new File("src/main/resources/xml");
        if (!xmlDir.exists()) {
            xmlDir.mkdir();
        }
        try {
            JAXBContext jc = JAXBContext.newInstance("com.etp.resumeg.resumeg.xmlpojo");
            Marshaller m = jc.createMarshaller();
            OutputStream os = new FileOutputStream(xmlFilePath);
            m.marshal(template, os);
        } catch (JAXBException e) {
            e.printStackTrace();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    } catch (ParserConfigurationException e) {
        e.printStackTrace();
    } catch (TransformerConfigurationException e) {
        e.printStackTrace();
    } catch (TransformerException e) {
        e.printStackTrace();
```

Şekil D.1. createXmlTest fonksiyonu

Ek-E

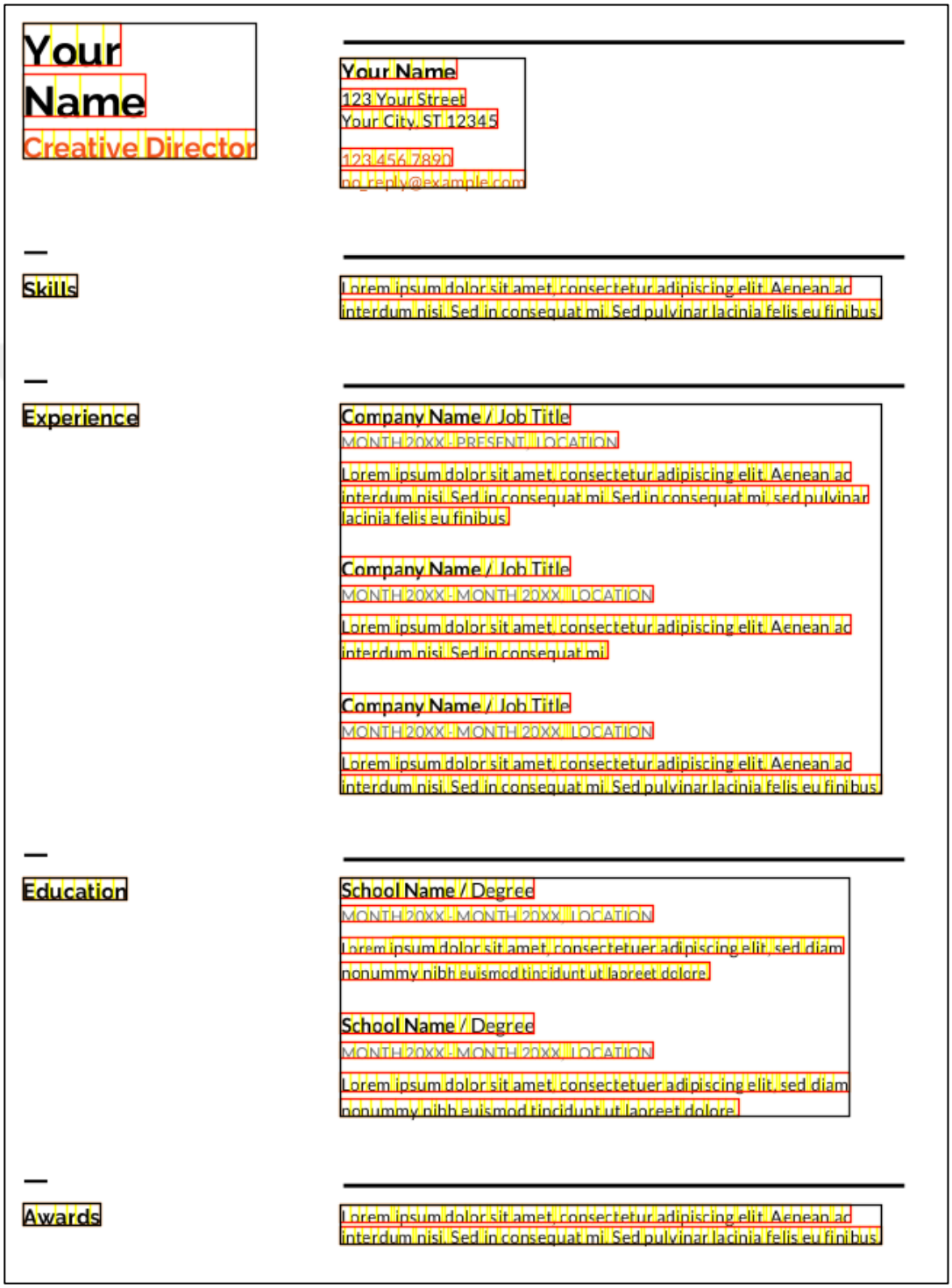
Şekil E.1'de CV şablon yapısı çıkartılmış örnek bir PDF gösterilmektedir. Görseldeki sarı renkli kutular karakterleri, kırmızı renktekiler satırları ve siyah renktekiler de paragraf yapılarını teşkil etmektedir. Ayrıca PDF 2 sütunlu bir yapıya sahip olduğundan, görselden satır ve paragrafların nasıl sütunlara göre gruplandırıldıklarıda görülebilir.



Şekil E.1. Şablonu çıkartılmış PDF Örnek1

Ek-F

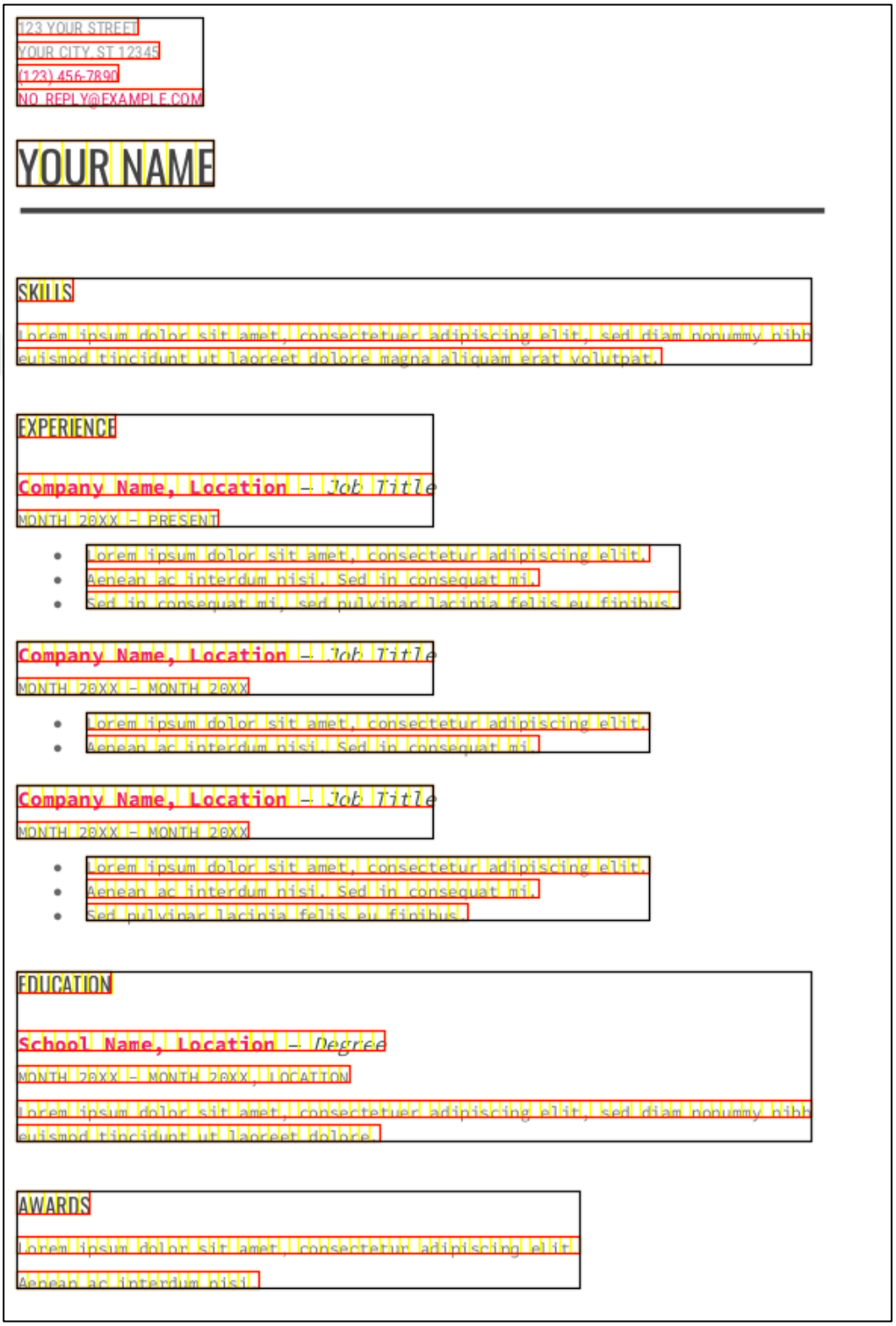
Şekil F.1'de CV şablon yapısı çıkartılmış örnek bir PDF gösterilmektedir.



Şekil F.1. Şablonu çıkartılmış PDF Örnek2

Ek-G

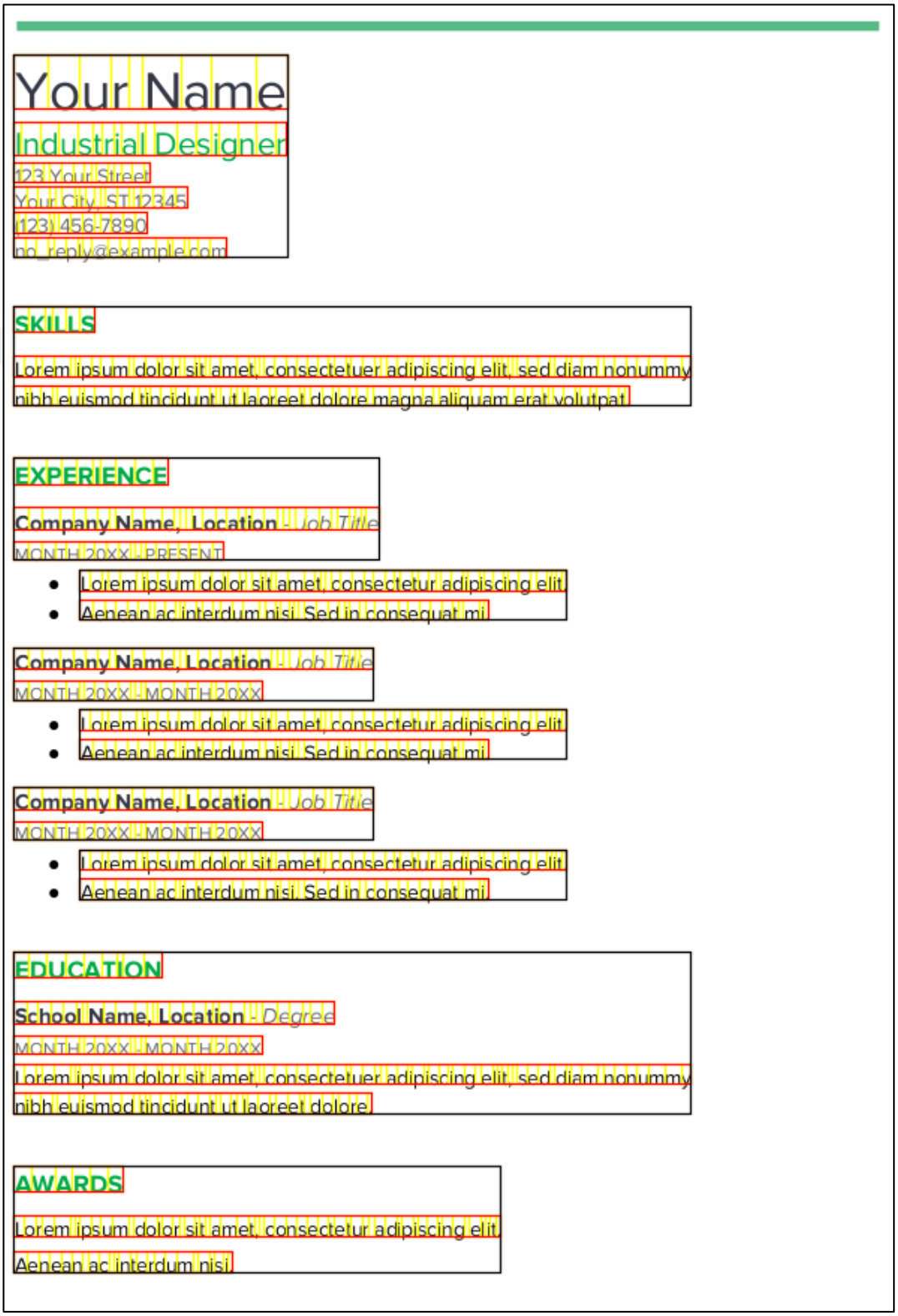
Şekil G.1'de CV şablon yapısı çıkartılmış örnek bir PDF gösterilmektedir.



Şekil G.1. Şablonu çıkartılmış PDF Örnek3

Ek-H

Şekil H.1'de CV şablon yapısı çıkartılmış örnek bir PDF gösterilmektedir.



Şekil H.1. Şablonu çıkartılmış PDF Örnek4

KİŞİSEL YAYIN VE ESERLER

Kantarci A., Eken S., Sayar A., Dijital Dokümanlar Üzerinde Otomatik Biçim Tanıma ve Farklı İçeriklere Uyarlama: Özgeçmişler Üzerinde Durum Çalışması, *Avrupa Bilim ve Teknoloji Dergisi*, 2019, (17), 1313-1324

ÖZGEÇMİŞ

Alper KANTARCI 1993'te Priştine'de doğdu. İlk okulu Prizren'in Emin Durak İlköğretim Okulu'nda okudu. Lise öğrenimini Gjon Buzuku Lisesi'nde tamamladı. 2012 yılında girdiği Kocaeli Üniversitesi Bilgisayar Mühendisliği Bölümü'nden 2017 yılında mezun oldu. Aynı yıl içinde Kocaeli Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı'nda yüksek lisans eğitimine başladı.