KOCAELİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

BİTİRME TEZİ

Veri Filtreleme ve Anlamlı Veri Oluşturma

Alican Akkuş

Abdurrahman Alkaya

Kocaeli Üniversitesi

DANIŞMAN: Doç. Dr. Ahmet SAYAR

KOCAELİ 2016

BİTİRME TEZİ SAVUNMASI

SINAV SONUCU JÜRİ ORTAK RAPORU

Eğitim-Öğretim Yılı: 2015 /2016 **Yarıyıl: Bahar**

Öğrencinin

No : 120202060 , **Adı, Soyadı :** Alican Akkuş

No : 120202076 , **Adı, Soyadı :** Abdurrahman Alkaya

Bitirme Tezi Başlığı : Data Filtreleme ve Anlamlı Veri Oluşturma Uygulaması

Bilgisayar Mühendisliğine,

Yukarıdaki başlıklı bitirme tezini tamamlayan Alican AKKUŞ'un tez savunması sınavı ..... / ..... / 201..... günü, saat .....**:**..... yapılmıştır. Savunma ..... dakika sürmüştür.

Adayın projesi hakkında **kabul / red / düzeltme** kararı **oybirliği / oy çokluğu** ile verilmiştir.

Açıklama :

Gereği için bilgilerinize saygılarımızla arz ederiz.

Tez Danışmanı	Jüri Üyesi	Jüri Üyesi
Doç Dr. Ahmet SAYAR	Yrd.Doç.Dr. Pınar Onar Durdu	Arş. Gör. Süleyman Eken

Bu dökümandaki tüm bilgiler, etik ve akademik kurallar çerçevisinde elde edilip sunulmuştur. Ayrıca yine bu kurallar çerçevesinde kendime ait olmayan ve kendiminüretmediği ve başka kaynaklardan elde edilen bilgiler ve metaryeller (text, resim, şekil, tablo vb.) gerekli şekilde referans edilmiş ve dökümanda belirtilmiştir.

Alican Akkuş

Abdurrahman Alkaya

ÖNSÖZ VE TEŞEKKÜR

Bitirme projesi kapsamında fikir ve görüşleri ile yol gösteren danışman hocam Sayın Doç. Dr. Ahmet SAYAR'a ve proje süresince yardımlarını esirgemeyen sevgili hocam Arş. Gör. Süleyman EKEN'e de teşekkürü bir borç biliriz.

Haziran 2016

Alican Akkuş

Abdurrahman Alkaya

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

VERI FILTRELEME VE ANLAMLI VERI OLUŞTURMA

ÖZET

Veri filtreleme ve Cümle temizleme kavramları, günümüzde oldukça artan dosya ve veri boyutları ile birlikte daha çok anlam kazanmıştır. Sadece Web üzerinde bir gün içerisinde binlerce GB'lık veri oluşmaktadır. Oluşan bu devasa büyüklükteki veriler her zaman bizim için anlamlı ve faydalı bilgiler içermeyebilir. Daha doğrusu bizim için anlamlı olan ve anlamsız olan veriler iç içe geçmiş olabilir. Projemizde, log dosyaları ve türevleri üzerinde kullanıcı için faydalı olarak tanımlanan verilerin işlenmesi üzerinde durulmuştur. Temel amacımız, kullanıcının ilgilendiği verileri diğer verilerden ayırmaktır.

Anahtar Kelimeler: Veri Filtreleme, Cümle İşleme, Anlamlı Veri Oluşturma

DATA FILTERING AND TEXT CLEANING

ABSTRACT

Nowadays, Data filtering and text cleaning concepts has become such more meaning with increasing size of files and data, only Thousands GB data is composed on the web in a day. Consisted colossal amount of data may not always meaningful and valuable information for us.Or rather, that is meaningful and meaningless data for us may be entwined. Our project is focused on the processing data that is described as useful on the log files and their derivations for the user. Our main purpose , seperated data that is user is interested from other data.

Keywords: Data Filtering, Text Cleaning, Text Marker, Text Analysis

GİRİŞ

Büyük veri günümüzde oldukça yaygın olarak kullanılan bir kavramdır. Büyük veri, medya paylaşımları, video, fotoğraf, log dosyaları gibi değişik kaynaklardan toparlanan tüm verinin anlamlı ve işlenebilir hale dönüştürülmesi biçimine denir. Günümüzde kullanılan ilişkisel veri tabanlarında işlenen yapısal verilerin aksine big data yapısal olmayan veri yığınıdır. Eskiden bilişim inanışı olan yapısal olmayan verinin değersizliği anlayışını yıkan sistemdir [1].

Günümüzde uygulamaların çalışma anındaki durumunu yansıtan log dosyalarının incelenmesi, olası sorunların önceden tespit edilmesi, uygulama performanslarının değerlendirilmesi oldukça önem kazanmaktadır.

Büyük log dosyalarının incelenmesi, yararlı ve kullanılabilir hale getirilmesi ile birlikte analiz, değerlendirme ve olası sorunların önceden tespiti gibi işlemler oldukça zor ve maliyetlidir.

Log dosyaları, geliştiriciler için kısmen anlamlı olmasına rağmen bilgisayarlar için oldukça anlamsızdır. Artan dosya boyutu ile beraber geliştiriciler için de oldukça anlamsız hale gelmektedir. DATA FILTERING AND TEXT CLEANING yani Log Processor uygulamamızın temel amacı, bu analizleri yapmak ve bilgisayarlar için tamamen anlamsız olan log dosyalarını anlamlı hale getirmek, anlamlı hale getirilen bilgi ve değerlendirmelerin kullanıcılara sunulmasını sağlamaktır.

Log processor uygulması temelde bir konfigürasyon üzerinden işlemlerini gerçekleştirir. Kendisine verilen konfigürasyon dosyası ile birlikte şunları yapabilme özelliklerine sahiptir;

● İşleme : Kullanıcının belirlediği template ile log dosyaları işlenir.

● Sunma : Kullanıcıya HTTP üzerinden işlenen veriler sunulur.

● Export : Kullanıcının belirlediği adreslere işlenen verilerin export edilmesini sağlar.

Tez dokümanı boyunca yukarıdaki kısımlar detaylandırılacaktır. Uygulama, text tabanlı tüm dosya tiplerini destekler.

Uygulamaın desteklediği bazı dosya formatları aşağıdaki gibidir;

● .log uzantılı dosyalar.

● .txt uzantılı dosyalar.

● .csv uzantılı dosyalar.

● Text tabanlı tüm dosyalar.

1. GENEL BİLGİLER

1.1. Tez Çalışmasının Amacı ve Başlatılma Nedenleri

Bilgisayar kullanıcı sayılarının artması ile birlikte uygulama kullanıcı sayılarında da artış meydana gelmiştir. İnsanların kullanımına sunulan uygulamaların, çalışma anında uygulamanın durumu hakkında çıktılar üretmesi, uygulama üzerinde yapılan işlemlerin gözle görülür olarak tutulması ihtiyaçları doğmuştur. Uygulama kullanıcı sayısı artıkça oluşan bu çıktıların boyutları da paralel olarak oldukça artmaktadır.Artan dosya boyutları ile beraber bu dosyaların analiz edilmesi, uygulama performansının değerlendirilmesi, kullanıcıyı ilgilendiren işlemlerde herhangi bir sorun olması durumunda incelenmesi gibi işlemler geliştiricileri ve sadece Log Analizi yapan çalışanları oldukça zorlamaktadır. Bu nedenle bazı tool'lar geliştirilmiş olup bu işlemlerin kolaylaştırılması amaçlanmıştır.

Projemizde, log ve benzeri text dosyalarının analiz edilmesi, değerlendirilmesi, gerekli görülürse export edilmesi gibi işlemlerin manuel olarak bir insan tarafından yapılmadan, uygulama ile yapılmasını amaçladık.

1.2. Tez Çalışmasının Katkıları

Bu tez kapsamında aşağıdaki çıktılar hedeflenmiştir:

- Bu çalışmada Log ve benzeri text dosyalarının yönetilmesinin kolaşlaştırılması sağlandı.
- Çalışma ile birlikte analiz edilen dosyalar, kullanıcılar, geliştiriciler ve bilgisayarlar için anlanmlı hale geldi.
- Proje kapsamında birçok open-source teknolojilerden yararlanılarak açık kaynak ekosistemine katkı sağlandı. Ayrıca proje dokumantasyonu ile birlikte open-source olarak yayınlandı.
- Bunların yanında kendimizin geliştirmiş olduğu Text Marker'lama ile farklı bir bakış açısı kazanıldı.

1.3. Tez Düzeni

Tezin 1.bölümünde tez çalışmasının amacı, başlatılma nedenleri ile birlikte tezin katkıları yer almaktadır. 2.bölümünde projemizin temelini oluşturan Log dosya analizleri ile ilgili yapılan çalışmalar yer almaktadır. 3.bölümde temel anlatımlar ele alınmıştır. Text markerlerlama, text extraction gibi konuları ele alınmıştır. 4.bölümde Log dosyalarının işlenmesine ve mimarisine değinilmiştir. Log dosyasının içeriği, analizlerin nasıl yapıldığı, projenin kod içeriğinin detaylı bir şekilde açıklanması, başarım değerlendirilmesi gibi konular bu bölüm altında ele alınmıştır. 5.bölümde ise sonuç ve öneriler kısmı yer almaktadır.

2. İLGİLİ ÇALIŞMALAR

2.1. Text Marker Çalışmaları

Literatürde yapılan text marker çalışmaları ; doğal dil işlemeye yönelik çalışmalardır. [2]. Literatürde yapılan text marker; bir syntax detector ile cümlenin analiz edilip parametrik olarak skip, replace gibi işlemler ile dil işleme, cümleden anlam çıkarsama, anlamlı veriyi anlamsız olandan ayrıma gibi işlemler yapmakta. Filter olarak sadece RegexMarker Filter seçeneği bulunmaktadır. Yani syntax detector ile bulduğu bir kelimeyi sadece regex ile işleme sokabilir. Toplamda iki dil için bunu yapabilmekte, İngilizce ve Fransızca.

2.2. Text Classification Çalışmaları

Text extraction ve sınıflandırma çalışmalarının merkezinde aranılan verinin diğer verilerden ayrılması bulunmaktadır. Terminolojide Classification gibi isimlendirmeler mevcuttur. Birçok farklı sınıflandırma methodu bulunmaktadır. Yapılan diğer bir çalışma da ise farklı bir method öne sürülmüştür [3].

Öne sürülen yeni method FKC olarak tanınlanmış olup Frequent Keyword Chain olarak detaylandırılmıştır. Bu çalışmada da stopSet, puncSet, suspectSet, denseSet gibi markerlar kullanılarak dokumanların sınıflandırılması gerçekleştirilmiştir. Çalışmanın temel amacı çok küçük boyutlarda da sınıflandırmayı text verisi üzerinden markerlama yaparak extract edip işleyebilmesidir.

Örneğin, bir email benzeri ifadeler boyut olarak küçük olmalarına rağmen önem derecesi yüksek olabilmektedir. İnput olarak alınan veri'de az miktarda kelime geçmesi FKC methodu için herhangi bir sorun oluşturmamaktadır.

2.3. Text Extraction Çalışmaları

Text extraction çalışmalarının merkezinde aranılan verinin diğer verilerden ayrılmasıdır. Terminolojide Regex, Extracting, Parsing gibi isimlendirmeler mevcuttur. Programlama dillerinin hemen hemen hepsinde yukarıda ifade edilen extraction, parsing gibi işlemler için hazır kütüphaneler ve api'ler yazılmıştır.

Parsing veya Syntax Analysis işlemleri doğal dillerde ve/veya bilgisayar dillerinde yapılabilmektedir. Parsing'in kökeni Latince'deki pars kelimesinden gelmektedir.

Yapılan tüm çalışmalar genel bir amaca yöneliktir. Regex, Parsing vb işlemler verilen bir input'a göre output üretirler. Genel amaçlardan bazıları, inputta herhangi bir kelime bulma, sub kelimeleri üretme, input'un belirli bir formata uyması(sadece alfabetik karekterler içermesi gibi) gibi genel amaçlar üzerinde durulmuştur. Spesifik olarak örneğin verilen input üzerinde, belirlenmiş bir template ile işlemler yapmak mümkün değildir.

3. VERİ FİLTRELEME VE ANLAMLI VERİ OLUŞTURMA

Bu kısımda yazılan uygulamaın mimarisi, akışı ve örnek çıktılarına yer verilecektir.

3.1 Log Dosyası Formatı ve İçeriği

Dosya formatı konusunda uygulama için herhangi text tabanlı bir formata sahip olunması yeterlidir. Dosya, stream olarak okunduğu için birçok formatı destekleyebilir.

- .log uzantılı dosyalar.
- .txt uzantılı dosyalar.
- .csv uzantılı dosyalar.
- Text tabanlı tüm dosyalar.

Log dosyası içeriği olarak uygulamaların runtime anında durumunu belirten her türlü bilgi olabilir.

Örnek bir log dosyası içeriği şöyle olabilir;

```
2016-01-06 18:57:30 DEBUG ContextListener - 06 Jan 2016 06:57:30 contextInitialized Log4j and Scheduler job initialized 126
```

Yaptığımız bu çalışma ile birlikte yukarıdaki log satırına benzer ifadeler içeren bir log dosyasının analiz edilmesi, değerlendirilmesi gibi işlemler yapıyor olacağız. Çalışma sonunda kullanıcı, uygulama performansını değerlendirebilir ve log'un anlamlı hale getirilmesini gözlemlyebilir.

3.2 Uygulama Konfigürasyon İçeriği

Uygulama bir konfigürasyon dosyası input olarak işlenecek olan log dosyasının, nasıl işleneceğini, işlenen verilerin nerelere aktarılacağını, nasıl sunululması gerektiği gibi bilgileri okuyarak çalışmasını sürdürür. Xml formatında bir dosyadır.

Konfigürasyon dosyasını kullanıcı istediği gibi düzenleyebilir, üzerinde oynama yapabilir. Konfigürasyon içeriği detaylıca açıklanmaya çalışılmıştır.

Database

Konfigürasyon dosyası içerisinde yer alan ve işlenen dosyanın memory database üzerinde geçici olarak saklanılmasını sağlar.

<database></database> etiketleri arasında yer alır.

Bu etiketler arasında tanımlanabilecek özellikler aşağıdaki gibidir;

- Driver : Memory driver implementasyonu için gerekli JDBC driver tanımı.
- URL : Memory database için base url tanımı.
- Username ve Passowrd : Memory daabase için username ve password bilgisi.
- Mode : Memory database'in hangi RMDS veritabı tipi gibi davranmasını gerektiği bilgisi.

Örnek bir database etiketi aşağıdaki gibidir;

```
<database>
        <param name="driver" value="org.h2.Driver" />
        <param name="baseUrl" value="jdbc:h2:mem" />
        <param name="dbName" value="bitirme" />
        <param name="dbUser" value="wora" />
        <param name="dbPassword" value="wora" />
        <param name="mode" value="Derby" />
</database>
```

Monitoring

Verilen dosyanın processor sonunu işlenmiş çıktısının nasıl sunulacağını belirtir. Konfigürasyon dosyasında monitorings etiketleri
arasında tanımlanır. Kullanıcı aynı anda birden fazla sunum katmanı oluşturabilir. Bu özellik, HTTP, HTTPS gibi farklı kullanımlar için
oluşturulmuştur.

Uygulama, startup ile birlikte embed bir application server oluşturularak LogProcessor.war uygulamasını deploy eder.
Deploy edilen uygulama, Log Processor'den alınan çıktıları kullanıcıya sunar.

Kullanıcı, konfigürasyon dosyasında monitoring için şu
tanımları kullanabilir;

- Port : Uygulamanın hangi porttan çalışacağını belirtir.
- ContextName : Uygulamanın web display name'ini belirtir.
- HTTPS : Uygulamanın http/https olarak erişilebileceğini belirtir.
- Keystore File ve Password : Https seçilmesi durumunda keystore bilgisinin girilmesini sağlar.

Örnek bir monitoring tanımı aşağıdaki gibidir;

```
<!-- one or more monitoring different port time at same time -->
<monitoring>
        <param name="contextName" value="/MyLogProcessor" />
        <param name="portNumber" value="10800" />
        <param name="https" value="true" />
        <param name="keystoreFile" value="web/conf" />
        <param name="keystorePassword" value="wora" />
</monitoring>
```

Destinations

Log Processor çıktılarının nereye aktarılacağını belirtir. Kullanıcı, aynı anda birden fazla destianions seçerek sonuçların birden fazla yere
aktarılmasını sağlayabilir.

Kullanılabilir olan toplam 3 adet destinations vardır. Bunlar, File, DB ve Socket destinations'dur.
Kullanıcı, aynı anda birden fazla destinations adresi tanımlayarak sonuçların birden fazla yere gönderilmesini sağlayabilir.

File Destinations

Log Processor çıktılarının local dosya sistemine aktarılmasını sağlar. Destinations tanımında class olarak
"com.wora.adaptor.FileAdaptor" değerinin girilmesi gerekir.

File Destinations için tanımlanabilecek özellikler şunlardır;

- File Name : Sonuçların yazılacağı dosya ismini belirtir.
- File Suffix : File Name'e ek olarak default timestampt bilgisi eklenir. Kullanıcı bunu değiştirebilir. Örneğin, dosya adı olarak
- MyLogProccessor seçildi ise oluşacak olan local dosya adı MyLogProccessor_21052016174550 olarak düzenlenir.
- Export Type : Local dosya sistemin sonuçların nasıl yazılacağını belirtir. Default olarak dosya tipi xml'dir.
- File Size : Verilen dosya boyutuna dosya ulaştığında yeni bir dosya oluşturarak sonuçları oraya export eder. Default değer 100
- mb'dir.
- File Locaitons : Dosyanın local dosya sistemindeki yolunu belirtir.

Örnek file destination tanımı aşağıdaki gibidir;

```
<destination id="fileDestination" class="com.wora.adaptor.FileAdaptor">
        <param name="fileName" value="MyLogAnalysis" />
        <param name="fileSuffix" value="DD-MMM-YYYY" />
        <param name="exportType" value="xml" />
        <param name="fileSize" value="100MB" />
        <param name="fileLocation" value="/home/wora/Analysis" />
</destination>
```

DB Destinations

Log Processor çıktılarının herhangi bir RMDS veritabanı sistemine aktarılmasını sağlar. Destinations tanımında class olarak
“com.wora.adaptor.DBAdaptor” değerinin girilmesi gerekir.

DB Destinations için tanımlanabilecek özellikler şunlardır;

- DB Driver : RMDS veritabanı için JDBC driver bilgisidir.
- DB Url : RMDS veritabanı için JDBC url bilgisidir.
- DB User ve Password: RMDS veritabanı için kullanıcı adı ve şifre bilgisini içerir.

Örnek DB destination tanımı aşağıdaki gibidir;

```
<destination id="dbDestination" class="com.wora.adaptor.DbAdaptor">
        <param name="dbDriver" value="com.postgresql.Driver" />
        <param name="dbUrl" value="jdbc:postgresql://localhost:5432/Analysis" />
        <param name="dbUser" value="wora" />
        <param name="dbPassword" value="wora" />
</destination>
```

Socket Destinations

Log Processor çıktılarının herhangi bir socket sistemine aktarılmasını sağlar.
Destinations tanımında class olarak
“com.wora.adaptor.SocketAdaptor” değerinin girilmesi gerekir.
SocketDestinations için tanımlanabilecek özellikler şunlardır;

- Server Adress : Server socket adresini belirtir. Default olarak 127.0.0.1’dir.
- Server Port : Server socket port bilgisidir. Default olarak 5003’dür.
- Buffer Size : Socket buffer size belirtir. Default olarak 2048kb’dir.
- Idle Time : Uygulama belirli aralıklarla socket bağlantısını kontrol eder. Verilen idle time süresi kadar herhangi bir hareketlilik yok
- ise socket bağlantısını kapatır.
- Delay : Log Processor çıktılarının gönderilme frekansını belirler. Default olarak 1000ms ile her 1 sn de bir çıktı sonucunu iletir.

Örnek Socket destination tanımı aşağıdaki gibidir;

```
<destination id="socketDestination" class="com.wora.adaptor.SocketAdaptor">
        <param name="serverAdress" value="127.0.0.1" />
        <param name="serverPort" value="5003" />
        <param name="bufferSize" value="2048" />
        <param name="idleTime" value="60" />
        <param name="delay" value="1000" />
</destination>
```

Destinations tanımları sona ermiştir.

Files

İşlenecek olan dosyayı belirtir. Files etiketleri arasına eklenir. Kullanıcı birden fazla dosya tanımı yapabilir. Tanımlanan her bir dosyaya
id attribute ile unique bir değer girilmelidir.
Örnek dosya tanımı aşağıdaki gibidir;

```
<files>
        <file id="myLog">
                <param name="fileName" value="MyLog" />
                <param name="fileLocation" value="hdfs://localhost:54310" />
        </file>
</files>
```

Templates

Verilen dosyaların nasıl işleneceğini belirten en önemli konfigürasyon tanımıdır. Log satırının neler içerdiği, tiplerinin ne olduğunu ve bu alanlar üzeride tanımlanmış formatları belirtir.

Templates içerisidne ve birden fazla template tanımlanabilir. Her
Template'in unique bir id'si olmalıdır. Kuıllanıcı id ile bilrlikte alias olarak template'e bir isim verebilir.

Örnek bir template aşağıdaki gibidir;

```
<template id="methodTemplate" name="MethodLog">
        <param name="startChar" value="2" />
        <param name="endChar" value="3" />
        <param name="delimeterChar" value="26" />
```

```
        <param name="dataLength" value="3">
                <param name="dateOfData" type="date" sequence="1"
description="log data date" format="dd-M-yyyy" pattern="less
than %05-12-2016%" />
                <param name="methodDesc" type="string" sequence="3"
description="method description" pattern="if like %starting
server%"/>
                <param name="methodRunningTime" type="integer" sequence="4"
description="method running time"
pattern="greater than %100%"/>
        <file name="myFile" source="myLog" />
        <destination name="DbDest" source="dbDestination" />
        <destination name="SocketDest" source="socketDestination" />
</template>
```

Template içerisinde kullanılan değişkenlerin açıklaması aşağıdaki gibidir;

● Start Char ve End Char : Analiz edilecek dosya içerisinde sadece ilgilenilen alanlara odaklanılması için kullanıcı start ve end
karekterini belirtir. Uygulama da bu karekterlere göre texti markerlar ve sadece markerlanan katekterler arasındaki veri ile ilgilenir. Uygulama verilen bu karekterler arasındaki değerlere odaklanır.

● Delimeter Char : Start ve End karekterleri arasında belirli bir ayıraçlarla ayrılmış datanın oldugunu ve verilen ayıraçla ayrıldğını
belirtir.

● Data Length : Ayıraçlarla ayrılmış datanın uzunluğunu ve bu ayıraçlardaki değerlerin tanımını yapar. Örnekte, bu değerin 43
oldugunu ve sırasıyla dosya da start ve end karekterleri arasında ayıraçla şu değerlerin yer aldığını belirtir;

1- dateOfData : İlk değerin date tipinde olduğunu belirtir. Kullanıcı bu alan üzerinde format tanımı yapabilir ve filtering, cleaning
tanımı olarak pattern kullanabilir. Format ve Pattern'a uymayan veriler işlenmeyecektir. Örnekte Date değerinin 05-12-2016
tarihinden küçükk olan verilen işlenmesi istenmiştir.

2- methodDesc : İkinci değerin, string tipinde bir method açıklamsı olduğu belirtilmiş ve "starting server" 'a benzedigi
belirtimiştir. Bu tanıma uyan veriler analiz edilebilecek demektir.

4- methodRunningTime : Üçüncü alan integer tipinde ve methodun çalışma zamanını belirten bir değer oldugunu belirtir. Bu
değerin 100ms'den büyük olması istenmiş.

● My File : Template'in işleyeceği dosya'ya işaret eder. Source alanı, konfigürasyon içerisindeki file tanımındaki id ile eşleşmelidir.

● DbDest ve SocketDest : Bu tanımlamalar, log processorun template'den aldığı sonuçları aynı anda DB ve Socket'e aktarılması
belirtilmiştir.

4. GELİŞTİRİLEN SİSTEMİN MİMARİSİ

Log Processor uygulamamızın mimarisi ve uygulama akışı anlatılacaktır.
Uygulamada kuıllanılan teknolojiler aşağıdaki gibidir;

Java, Tomcat, H2 Memory Database, Apache.

4.1 MİMARİ

Uygulama, input olarak sadece bir konfigürasyon dosyasını alır. Alınan bu dosya ile birlikte uygulama hangi dosyayı nasıl işleyip sunacağını ve nasıl export edecegine karar verir.

Uygulama, configürasyon dosyası ile hangi dosyayı, nasıl işleyeceğini ve nasıl sunacağını anlamaktadır.

Sunum katmanı, embedded Tomcat7 ile oluşturulmaktadır. Uygulama ayağa kalktığında embed bir Tomcat server oluşturur ve sunum katmanında kullanılacak olan uygulamayı buraya deploy eder.

Kullanıcı, template'ler oluşturarak hangi dosyanın nasıl işleneceğine karar verme imkanına sahiptir. Örneğin, kullanıcı A dosyasının işlenmesini istemektedir, A dosyası içerisinde "server shutdown!" kısımlarını işlemek isteyebilir, işlemin sonucunun HTTP üzerinden sunulmasını sağlayabilir. Ayrıca bu işlemin sonucunu dosyaya, database'e veya socket'e aktarılmasını isteyebilir.

Kullanıcı, uygulama sayesinde örnekte ifade edildiği gibi A dosyası içerisinde "server shutdown!" ifadesinin ne sıklıklarla, kaç kere geçtiğini bulabilir, büyük dosyalarda manuel olarak arama zahmetinden kurtulabilir.
A dosya örneği, uygulamanın Data Cleaning özelliğine işaret eder.

Bir diğer örnek ise B dosyasında "server shutdown!" ifadesinde server'ın kapanma süresinin 500ms'den büyük olan kısımlarla ilgilenebilir. Sonuç olarak, kullanıcı B dosyasında yer alan ve "server shutdown!", 500ms'den büyük olduğu durumları işlemesini sağlar.
Bu ise uygulaman Data Filtering özelliğine işaret eder.

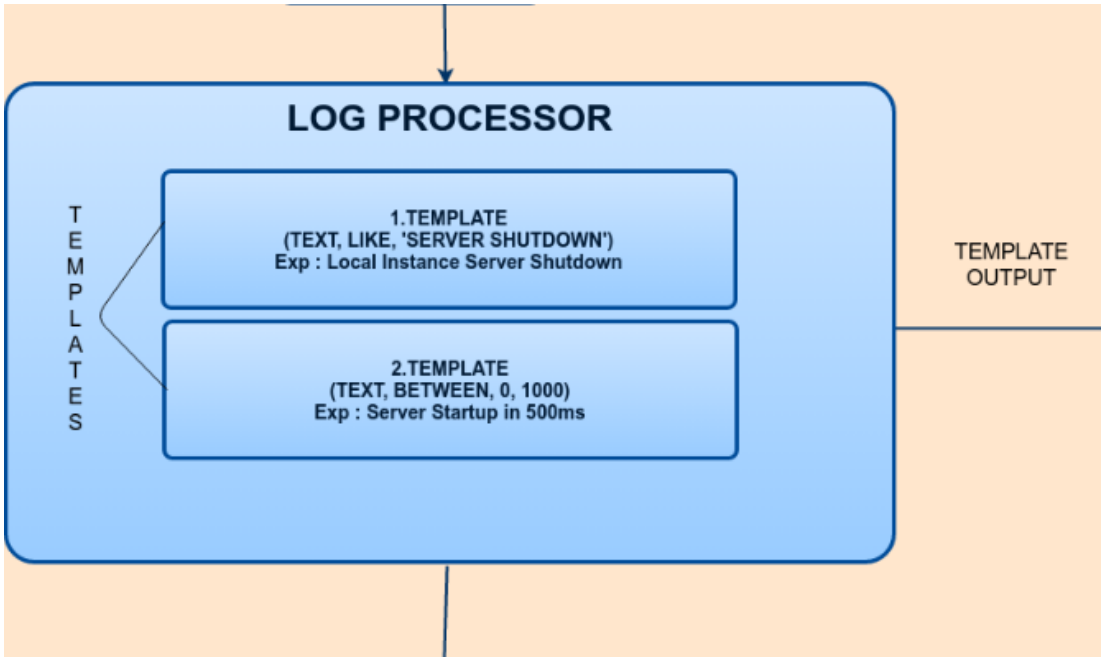
4.2 UYGULAMA AKIŞI

Uygulama mimarisi, temel olarak 3 ana kısımdan oluşmaktadır. Bunlar, Log Processor, Visualization ve Adaptor'dür.

Uygulama kendisine input olarak verilen dosyaları yine kendisine verilen template ile işler ve visualiazation ve adaptor katmanlarına iletir.
Bu katmanlar, kendisine iletilen bilgileri kullanıcıya ulaştırmaktan sorumludur.

Bu katmanlar aşağıda detaylandırılmıştır.

Log Processor Katmanı;



Şekil 1. Log İşleme Katmanı

Log Processor katmanı, kendisine verilen konfigürasyon dosyası içerisinde yer alan log dosyasını, tanımlanan template'lere göre işler. Template'ler içerisinde tanımlanan kontrollere, kısıtlamalara ve logic'lere göre template'den bir output çıktısı oluşur. Oluşan bu output Processor tarafından memory database de saklanır. Output ayrıca sunum katmanına da iletilerek templete processing işleminin sonucu görüntülebilir duruma gelir.

Bu katman ilgili log satırının tanımlanan template'e uyup uymadığını inceler. Tez içerisinde vermiş olduğumu örnek template(method template) içerisinde tarih, string ve numeric validasyonlar kullanılmıştır. Buradaki, uygulama kullanıcısına geniş bir alan sağlayarak istediğini yapabilmesini amaçlıyoruz.
Örneğin; Tarih olarak 09-06-2016 tarihden itibaren ve "startingg server" ibaresi içeren satırları ayıklayabilir.

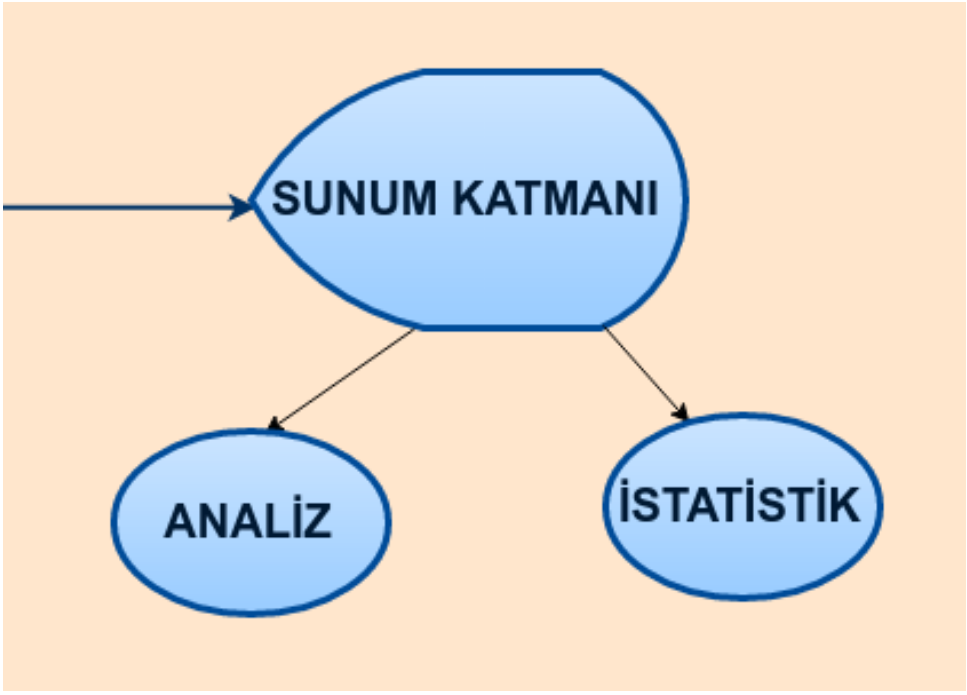
Yine bir diğer örnekte, "server shutdown" ifadesi geçen satırlada numeric ifade olan shutdown süresinin 100ms'den büyük olduğu durumlar irdelenebilir. Bu sayede uygulama kullanıcısı, server'ın ne zaman hangi performansla kendisini kapatabildiğini incelemiş olacabilecektir.

Örnek template'ler şu şekilde olabilir;

- Log'da server shutdown ifadesi geçen satırlar ile ilgileniyorum, sadece bu alanlar analiz edilsin.
- Log'da server started ifadesi geçen satırlar ile ilgileniyorum, sadece bu alanlar analiz edilsin.
- Log'da server shutdown ifadesi içerisinde yer alan çalışma zamanı olarak 100ms'den büyük olan satırlar ile ilgileniyorum, sadece bu alanlar analiz edilsin.
- Log'da server started olarak tarih özelinde 09-06-2016 tarihinden sonraki kısımlar ile ilgileniyorum, sadece bu alanlar analiz edilsin.

Bu örnekler çoğaltılabilir.

Sunum katmanı;



Şekil 2. Sunum Katmanı

Sunum katmanı, HTTP ve/veya HTTPS olarak ayarlanabilir. Aynı anda birden fazla kanalda monitoring yapılabilir, ssl / port gibi bilgiler kullanıcı tarafından kolayca konfigüre edilebilir.

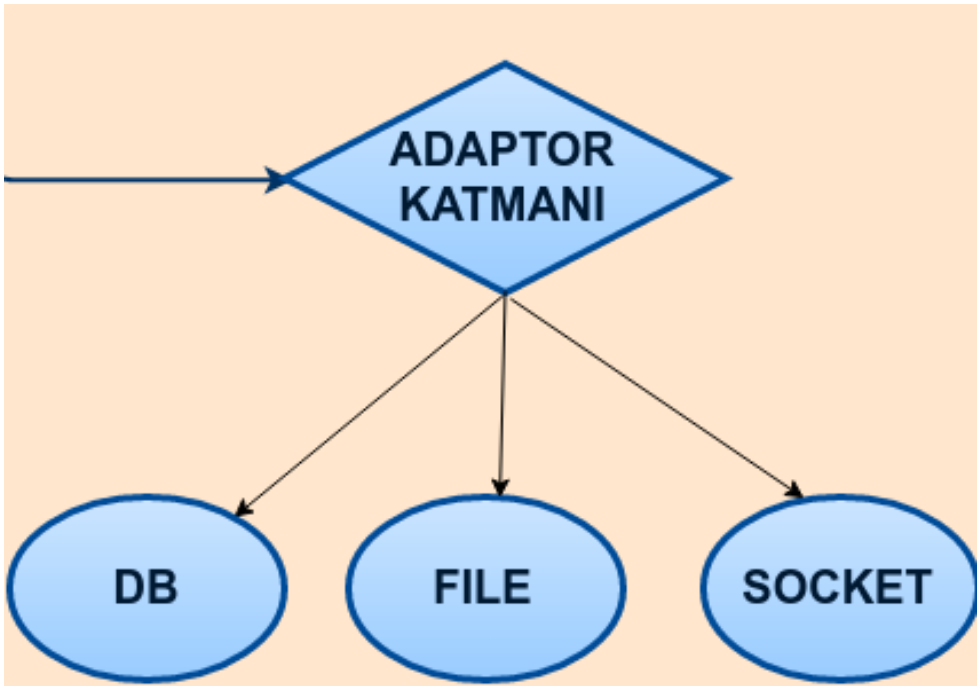
Analiz edilen verilerin memory database'e eklenmesi ile bu verilerin sunum ve analiz katmanına iletildiğini ifade ettik. Ayrıca bu veriler, tanımlanan export adreslerine aktarılması için de Adaptor katmaına da iletilirler. Adaptor katmanı, kendisine verilen verileri tanımlanana adreslere iletmekle sorumludur.

Sunum katmanı, template içerisinde tanımlanan alanlardan faydalanarak memory db üzerinde bulunan verileri gösterir. Memory db içerisinde ki tablo ve sütun bilgileri, template'de tanımlanan değerlere göre otomatik olarak runtime anında oluşturulur. Örneğin yukarda vermis oldugumuz template'e göre uygulama aşağıdaki scripti oluşturacaktır. Memory üzerinde çalıştırmadan önce böyle bir tablonun varlığını sorgulayıp yok ise tabloyu oluşturmaktadır;

```
Create script : CREATE TABLE METHODTEMPLATE (
        ID int not null GENERATED ALWAYS AS IDENTITY (START WITH 1,
                INCREMENT BY 1) ,
        dateOfData date,
        methodName varchar,
        methodDesc varchar,
        methodRunningTime integer
)
```

Tabloda oluşturulan sütunlar, sunum katmanında kullanılacaktır. Uygulama bu verilerin dışında metada olarak, örneğin bir template'e uyan kaç adet verinin olduğunu, bunların sıklığını da sunum katmanında kullanıcıya göstermektedir.

Adaptor katmanı;

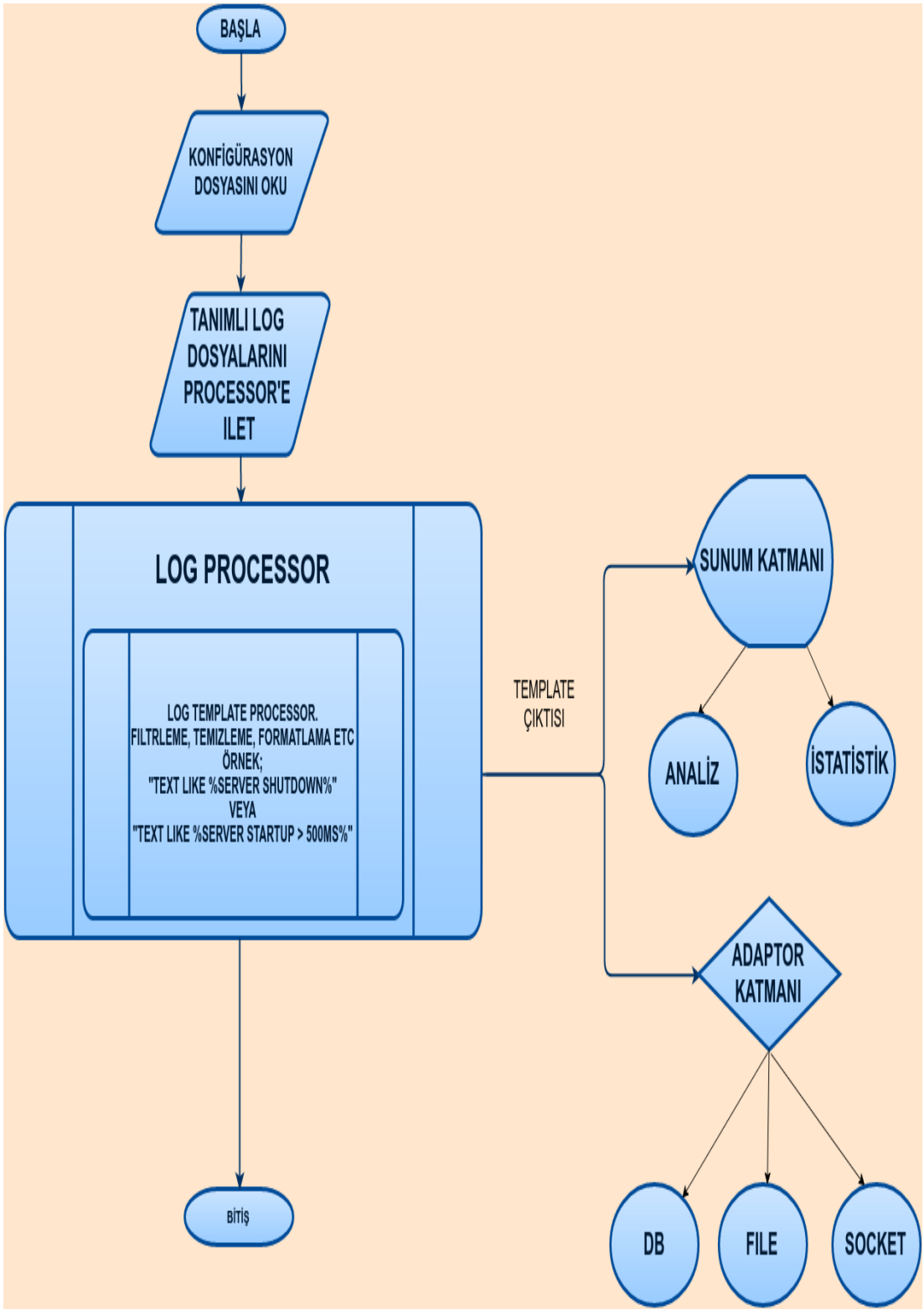


Şekil 3. Adaptör Katmanı

Log Processor'den gelen veriler Adaptor katmanına iletilir. Template'e assing edilmiş adaptorlerin tamamına bu veriler yine template'de belirlenen zaman sıklıkları ile aktarılır.
Her adaptor tanımı kendi başına çalışır, processor sadece ilgili adaptor'un mesaj kuyruğuna iletilecek olan mesajı ekler. Adaptorler, Thread olarak arka planda çalışır ve sürekli mesaj kuyruğunda "bana gelecek mesaj var mı?" diye kontrol eder.

Kendisine gönderilmiş bir mesaj var ise adaptor tipine göre DB, FILE yada SOCKET aracılığıyla hedef sisteme aktarır.

Mimarinin tam şekli ise şu şekildedir;



Şekil 4. Genel Mimari

4.3 TEMPLATE OLUŞTURMA

İlk olarak template nasıl programatik olarak oluşturuluyor ona bakalım;

```
private void createTemplates(Document document) throws Exception {

        Element templatesElement = XmlUtils.findNode(document, "//templates");
        NodeList templateNodes = XPathAPI.selectNodeList(templatesElement, "template");
        logger.info(templateNodes.getLength() + " template found.");

        for (int i = 0; i < templateNodes.getLength(); i++) {

                Element templateElement = (Element) templateNodes.item(i);
                LogTemplate logTemplate = new LogTemplate();
                logTemplate.setTemplateId(templateElement.getAttribute("templateId"));

                logTemplate.setTemplateName(templateElement.getAttribute("name"));

                …..

                ….

                …..

                templatesPool.put(logTemplate.getTemplateId(), logTemplate);

        }

}
```

Uygulama, configürasyon dosyası içerisindeki <templates> etiketi içerisindeki <template> etiketlerini okuyarak log template oluşturur. Kullanıcı aynı anda aynı log dosyası için farklı templateler oluşturabilir.

Her Template'in bir uniqe id'si bulunmak zorundadır. Oluşturulan template'ler bir Template Pool içerisinde saklanır. LogTemplate, Thread sınıfından kalıtılan bir Java class'ıdır.

Daha sonra template pool içerisindeki tüm template'ler Thread olarak start edilmektedir.

4.4 ANALİZ EDİLECEK DOSYANIN OLUŞTURULMASI

Analiz edilecek olan dosyaları, configürasyon dosyası içerisinde ki file etiketleri ile tanımlanır.

```
private void createFiles(Document document) throws Exception {

        Element filesElement = XmlUtils.findElement(document, "//files");
        NodeList files = XPathAPI.selectNodeList(filesElement, "file");
        logger.info(files.getLength() + " found in a configüration.");


        for (int i = 0; i < files.getLength(); i++) {

            Element element = (Element) files.item(i);
            LocalFile localFile = new LocalFile();
            localFile.setId(element.getAttribute("id"));
            NodeList params = XPathAPI.selectNodeList(element, "param");
            .....

            .....

            .....

            logger.debug(localFile.getId() + " log file definitions is created.");

            filesPool.put(localFile.getId(), localFile);

        }

    }
```

Uygulama, configürasyon dosyası içerisindeki <files> etiketi içerisindeki <file> etiketlerini okuyarak log dosyalarının bilgisini oluşturur. Kullanıcı aynı anda aynı birden fazla log dosyası tnaımı yapabilir.

Her Log file'ın bir uniqe id'si bulunmak zorundadır. Oluşturulan log dosyaları bir Files Pool içerisinde saklanır. Log Template, oluştururken bu pool içerisindeki dosyaları kullanır.

4.5 ADAPTOR KATMANI İÇİN TANIMLAMALAR

Analiz edilen çıktıların hedef sisteme aktartılmasını sağlayan adaptor tanımı aşağıdaki gibidir;

```
private void createDestinations(Document document) throws Exception {

                Element destinations =  XmlUtils.findNode(document, "//destinations");
                NodeList destinationsNodes = XPathAPI.selectNodeList(destinations,
"destination");
                logger.info(destinationsNodes.getLength() + " destinations found.");

                for(int i = 0; i < destinationsNodes.getLength(); i++){

                        Element destination = (Element) destinationsNodes.item(i);
                        String destionationID = destination.getAttribute("id");

                        AbstractAdaptor adaptor =
(AbstractAdaptor)Class.forName(destination.getAttribute("class")).newInstance();
                        adaptor.init(destination);

                        logger.debug(destionationID + " destination is created.");
                        adaptorsPool.put(destionationID, adaptor);

                }

        }
```

Uygulama, configürasyon dosyası içerisindeki <destinations> etiketi içerisindeki <destination> etiketlerini okuyarak log dosyalarının işlenmesi sonucu oluşan çıktıların nereye export edileceği bilgisini oluşturur.

Abstract Adaptor, kendisine verilen adaptor type için bir instance oluşturur ve initialize eder. Kullanılabilir adaptor tipleri ise DB, FILE ve SOCKET'DIR.

Kullanıcı aynı anda aynı birden fazla export tanımı tanımı yapabilir.

Her Destinatipn'ın bir uniqe id'si bulunmak zorundadır. Oluşturulan destinations'lar bir Adaptor Pool içerisinde saklanır. Log Template oluştururken bu pool içerisindeki adaptorlere bakarak kendisine atanmış olan adaptorleri kullanır.

4.6 SUNUM KATMANININ OLUŞTURULMASI

Uygulama, sunum katmanına verileri aktarabilmek için startup ile birlikte embed olarak bir Tomcat7 oluşturur ve server'a Monitoring uygulamasını deploy eder.

Monitoring uygulaması, Java Web ile yazılmış olup war dosya formatı ile paketlenmiş bir dosyadır. Kullanıcı, template ile tanımlamış olduğu bilgileri ekran üzerinde HTTP/HTTPS olarak görebilir, analiz sonuçlarını da karşılaştırabilir.

```
private void startMonitoring(Document document) throws Exception {

                logger.info("Starting web server...");
                tomcat = new Tomcat7();
                tomcat.initialize(document);
                tomcat.startTomcat();
                logger.info("Web server started.");
}
```

Embed Tomcat instance oluşturulur ve initialize edilerek start edilir.

```
private void initialize(Document document) throws Exception {
        Element monitorings = XmlUtils.findNode(document, "monitorings");
        NodeList monitoringList = XPathAPI.selectNodeList(monitorings, "//monitoring");
        logger.debug(monitoringList.getLength() + " monitoring found..");
        catalinaHome = document.getProperty("monitor.web.root", "web");
        String context = document.getProperty("monitor.web.context." + contextCounter + ".path");
        monitorPort = Integer.parseInt(document.getProperty("monitor.port", "8080").trim());
        httpsKeyStore = document.getProperty("monitor.https.keystore");
        useHttps = document.getProperty("monitor.use.https", "false").equalsIgnoreCase("true");
        String realmDigest = document.getProperty("monitor.realm.digest", "").trim();
        // // set the memory realm
        realmBase = new MemoryRealm();
                if (org.apache.commons.lang.StringUtils.isNotBlank(realmDigest)) {
                realmBase.setDigest(realmDigest.toUpperCase());
                }
}
```

Ayrıca Monitoring üzerinde authantication ve autharization işlemleri yapılabilmektedir.

Tomcat-user.xml dosyası içerisine kullanıcı rolleri oluşturulabilir, rollere sahip kullanıcı tanımları yapılabilir.

Default olarak monitoring ekranına giriş yapmak için username/password olarak admin/admin kullanıcı kullanılmalıdır.

4.7 LOG PROCESSOR İLE DOSYANIN İŞLENMESİ

Log Processor katmanı, kendisine verilen template ile dosyayı okur. Her bir satırı yardımcı katmanlar ile validate etmeye yönelik işlemler yapar.

Aşağıdaki kod bloğu, Processor'e verilen dosya ile template'i valide etmedeki ilk adımdır;

```
@Override
public void run() {

	BufferedReader br = null;
	try {

		String sCurrentLine;
		br = new BufferedReader(new FileReader(file.getFileLocation()));
		while ((sCurrentLine = br.readLine()) != null) {
				//if is valid line then add to internal memory
			String validScript = test(sCurrentLine);
			if(validScript != null){
				logger.debug("İnsert script : " + validScript);

				Analys.getInstance().getDbService().insertDataToMemory(validScript);
				for(AbstractAdaptor assignedAdator : adaptors){
					assignedAdator.addQueue(validScript);
				}

			}
		}

	} catch (Exception e) {
		logger.error(e, e);
		} finally {
			br.close();
		}

	}
```

Gelen verinin template'e uyup uymadığını test methodundaki validasyonlara bakılarak karar verilir;

```
public String test(String data) throws Exception {

        try {

        logger.info("Proccessing data : " + IOUtils.convertLogMessageToString(data));

        char startOfText =
IOUtils.getAsciiByValue(String.valueOf(params.get("startChar")));
        char endOfText = IOUtils.getAsciiByValue(String.valueOf(params.get("endChar")));
        char delimeter = IOUtils.getAsciiByValue(String.valueOf(params.get("delimeterChar")));

logger.info("startOfText : " + (int) startOfText + " -- endOfText : " + (int) endOfText + " -- delimeter :
" + (int) delimeter);

int startIndex = data.indexOf(startOfText), endIndex = data.indexOf(endOfText);

if(startIndex == -1 || endIndex == -1){
return null;
}

data = data.substring(startIndex + 2, endIndex);
String[] columns = data.split(String.valueOf(delimeter));
LinkedList<LineBean> lineBeans = this.getDataLine();

StringBuilder insertScript = new StringBuilder("insert into ");

insertScript.append(this.getTemplateName()).append("(");
        for (LineBean line : lineBeans) {

                insertScript.append(line.getName()).append(",");

        }

        insertScript.delete(insertScript.lastIndexOf(","), insertScript.length());
        insertScript.append(") values(");

boolean valid = true;
```

```
		for (int i = 0; i < columns.length; i++) {
			String column = columns[i];
			LineBean bean = dataLine.get(i);

			logger.debug(bean);
			valid = TemplateValidationProcessor.validdation(column, bean);

			if (!valid) {
				return null;
			}else{
				insertScript.append("'").append(column).append("',");
			}

		}

		insertScript.delete(insertScript.lastIndexOf(","), insertScript.length());
		insertScript.append(")");

		return insertScript.toString();

		} catch (Exception e) {
			logger.error(e, e);
		}


		return null;

}
```

TemplateValidationProcessor ise kendisine verilen value değer ile template'in uyup uymadığını kararlasıtır;

```
public static boolean validdation(String value, LineBean bean) throws Exception {


        switch (bean.getType()) {
        case "date":
                return validateDate(value, bean.getPattern(), bean.getFormat());
        case "string":
                return validateString(value, bean.getPattern());
        case "integer":
                return validateNumber(value, bean.getPattern());
        default:
                break;
        }
        return true;
}
```

Template üzerinde date, stirng ve numeric alanlar üzerinde validasyonlar yapılabilir. Date üzerinde yapılabilecek validasyonlar, less than, greather than gibi verilen tarihten önce yada sonrası için validasyon oluşturulabilir.

Yine string üzerinde değerin, verilen değerle eşleşmesi veya benzeşmesi gibi validasyonlar yapılabilir. Numeric olarak ise büyüktür ve küçüktür işlemleri yapılabilir.

Configürasyon içerisinde tanımlanabilecek date validasyomları için örnekler;

Pattern pattern=*"greather than %05-12-2016 11:45:00.000%"*

Pattern pattern=*"gt %05-12-2016 11:45:00.000%"*

Pattern pattern=*" > %05-12-2016 11:45:00.000%"*

Yukarıdaki örnekte log dosyasındaki tarihin 05-12-2016 tarihinden büyük olması istenmiştir. "greather than" , "gt" , ">" syntaxları aynı amaçla kullanılabilir.

Pattern pattern=*"less than %05-12-2016 11:45:00.000%"*

Pattern pattern=*"lt %05-12-2016 11:45:00.000%"*

Pattern pattern=*" < %05-12-2016 11:45:00.000%"*

Yukarıdaki örnekte ise log dosyasındaki tarihin 05-12-2016 tarihinden küçük olması istenmiştir. "less than" , "lt" , "<" syntaxları aynı amaçla kullanılabilir.

String ifadeler için validasyomlara örnek ise;

Pattern pattern=*"like %starting server%"*

Pattern pattern=*"eq %starting server%" yada* pattern=*"equals %starting server%"*

Validasyonun başarılı olması durumunda ilk olarak sunum ve diğer katmanlara iletilmesi için geçici olarak memory db üzerinde veri kaydedilir.

Kaydedilen veri, template'e assing edilmis olan adaptorlerin mesaj kuyruğuna eklenerek adaptor katmaıına da iletilmis olur.

5. SONUÇLAR VE ÖNERİLER

Günümüzde log analizi için birçok tool bulunmaktadır. Bu tool'ların genel amacı log takibini efektif bir şekilde yapmaktır. Ancak hiçbirinde kullanıcı konfigürasyonu üzerinden log dosyası üzerinde kriterler uygulayarak validasyonlar, sorgulamalar yapmak bulunmamaktadır.

Günümüzde sadece bu alanda ürünler çıkarılmakta ve bu işi yapacak çalışan aranmaktadır. Log takibi ve analizi yapmak isteyen kurumlar, bu ürünnleri ya satın almakta yada çalışan istihdam etmekteler. Bu da kurumlar için bir maliyet olmaktadır.

Yapmış olduğumuz çalışma ile Log dosyalarının analizinden, değerlendirilmesine, export edilmesine kadar geniş bir alanda bunu kullanıcı isteklerine göre yapmayı amaçladık.

KAYNAKÇA

[1] https://tr.wikipedia.org/wiki/B%C3%BCy%C3%BCk_veri

[2] https://github.com/urieli/talismane

[3] Identification of Deliberately Doctored Text Documents Using Frequent Keyword Chain (FKC) Model*

[4] http://www.editplus.info/wiki/Writing_a_text_filter

[5] http://www.alchemyapi.com/api/keyword/textc.html

[6] LogMaster: Mining Event Correlations in Logs of Large-scale Cluster Systems

[7] Fast Entropy Based Alert Detection in Super Computer Logs

[8] A Lightweight Algorithm for Message Type Extraction in System Application Logs

[9] Investigating Event Log Analysis with Minimum Apriori Information

[10] Storage and Retrieval of System Log Events using a Structured Schema based on Message Type Transformation

[11] Identifying User Behavior by Analyzing Web Server Access Log File

[12] An Evaluation Study on Log Parsing and Its Use in Log Mining

ÖZGEÇMİŞ

1994 Batman'da doğumdu. İlk okul eğitimi burada tamamladı. Daha sonra orta ve lise eğitimin İstanbulda aldı. 2012 yılında Kocaeli Üniversitesi Bilgisayar Mühendisliğini kazandı ve halen devam etmektedir. 1.5 senedir sektörde yazılım geliştirici olarak görev almaktadır. Java ve ilgili teknolojilerini severek kullanır, kişisel bloğunda teknoloji üzerinde yazılar yazmaktadır.

Alican Akkuş.

1991yılında İstanbul'da doğdu. İlköğretim ve lise eğitimini İstanbul'da tamamladı. Daha sonra 2010 yılında Samsun 19 Mayıs Üniversitesi Fen Bilgisi Öğretmenliğini kazandı.Bir dönem okuduktan sonra okulunu bıraktı. Ardından tekrar hazırlanıp 2012 yılında Kocaeli Üniversitesi Bilgisayar Mühendisliğini kazandı ve devam etmektedir. Java web teknolojileri ile ilgilenmektedir ve kendini geliştirme aşamasındadır.

Abdurrahman Alkaya