Screening the Candidates in IT Field Based on Semantic Web Technologies: Automatic Extraction of Technical Competencies from Unstructured Resumes

Mihaela-Irina ENĂCHESCU
Bucharest University of Economic Studies, Bucharest, Romania
irina.enachescu@csie.ase.ro

While trying to fill in empty positions in a short time frame, struggling to find the best candidates while competing with other recruiters for them, nowadays, HR personnel need to consider innovative ways for reaching faster the IT professionals. Manually searching across professional networks is no longer an option. This study introduces the prototype of a system that automatically screens the candidates in the IT field. Its main goal is to provide a valuable support in the first stage of the personnel selection by decreasing the number of errors that can occur when thousands of CVs/profiles are manually filtered to pick candidates for an interview. Our proposed system consists in a mobile application that automatically selects online profiles from professional websites (like Indeed, LinkedIn, Monster) and ranks them, to finally display the eligible candidates for a particular open position to the recruiter. We developed an ontology to support the matching between the knowledge in the candidate's resume and the requirements in the job description. While developing the ontology our primary focus was on the skills that are encompassed in a resume, as these are the key abilities when searching for the ideal candidate. The knowledge a job seeker should possess, respectively a job description requires, is divided in the following categories: programming languages, databases, frameworks, integrated development environments, methodologies and operating systems. First part of the implementation, automatically extracting the skills from unstructured resumes, was achieved using Apache Tika and GATE.

Keywords: *eRecruitment, Human Resource Ontology, Resume Screening, Semantic Web*

1 Introduction

In an industry field where the demand has certainly surpassed the offer, like information technology (IT), recruiters need to use all the means to reach the candidates with high potential. Wasting time to manually search across the candidates' profiles in professional networks increases the risk of losing candidates in behalf of companies that perform an automated screening.

According to [1] the individuals are using nowadays the opportunities the internet provides to introduce themselves and their activities to a broad wide audience. Given the large variety of networking platforms, it is a common approach for people to share personal content on more than one platform, having thereby multiple profiles, and even connecting them via hyperlinks.

While trying to fill in empty positions in a short time frame, it is clear that the manual screening of resumes in one or more professional websites (like LinkedIn, Indeed etc.) by the HR personnel can be considered an inefficient and prone to error practice. As stated in [2], the adoption of eRecruitment is about being able to attract the right candidate, while you are reducing both the hire costs and the average time to fill a vacancy.

[3] suggests that when searching for the individual that best fits a job ad, a recommender system should cover three different type of fits: person-job, person-team and person-organization. Usually only the first kind of matching is addressed through the available eRecruitment platforms, the other two being mostly evaluated in a face-to-face interview.

However, while admitting the benefits brought by the recommender systems, we should bear in mind that we need to avoid what Eli Pariser is describing as *The Filter Bubble* [4], referring to how the personalized

DOI: 10.12948/issn14531305/23.4.2019.05

search can narrow our worldview. He describes in [4] a Web based on the relevance idea, and draws attention to the fact that while we are enjoying an experience especially tailored for us in the Internet, we are being shown what the web thinks we want to see, but which is not necessarily what we need. Considering this aspect, we want to propose a system that helps the companies to reach as quickly as possible the candidates with high potential, supporting them to automate the screening phase of the selection process. Next steps of the selection (in detail review of the resume, interview and so on) are still performed by the HR personnel, where they must ensure the candidate has the experience described in his profile and is fitting into the company culture.

The rest of the paper is organized as follows. The next section presents related work in the field of eRecruitment for resume screening. Section 3 introduces the architecture of a prototype system used to automatically screen the candidates in IT, based on the use of Semantic Web technologies. In section 4 we describe a model for the ontology we developed to represent the skill-set of the candidate and the job offer requirements. Section 5 exposes the tools we used in order to automatically extract technical competencies from unstructured resumes. Finally, section 6 presents the conclusions of our research and suggest some future work steps.

2. Related Work

The rapid growth of the IT professionals market face the recruiters with new challenges. They are under pressure to hire quickly qualified candidates, while competing with other companies to reach first the skilled professionals. They cannot afford to miss eligible profiles and this is the main reason more and more research studies are focused on developing systems that help the HR employees to effectively screen across a significant amount of resumes. This section describes some of the systems that proven good results for shortlisting candidates for an open position.

In [5] the authors presented an intelligent tool for screening candidates' resumes, named EXPERT, that it is based on ontology mapping. The proposed system consists of three phases: collecting resumes in various formats and representing them in ontology documents, constructing from the job opening a job criteria ontology (with mandatory and optional requirements) and, finally, performing the mapping between the two constructed ontologies and selecting the eligible candidates. A similarity function was proposed to compute the matching and a threshold can be selected in order to adjust the minimum accepted compatibility ratio. The system was tested with about 500 CVs downloaded from different job portals and obtained high precision and recall measures, showing a 90% accuracy.

The authors of [6] explored a semantic approach to find and rank experts of research areas in computer science and engineering field, based on the information available about their work in web databases. Firstly, they designed an ontology and then constructed a knowledge base with the ontological representation of the academic information available about researchers. Further they built an academic social network with all the publications related to a given topic and the author – co-author relationships that exist for these publications. Finally, when searching for experts in a particular domain they retrieved a ranked list based on importance scores. In order to measure the score, they relied on the previous constructed network to determine how many scientific papers the researchers have written as first author or co-author, how their contributions are related to the given topic, and relationships with other candidates in the network. The authors evaluated the proposed model against human judgements and obtained good results.

Likewise, authors of [7] proposed an approach employing semantic technologies. They introduced a levelized taxonomy of skills, following an inheritance relationship. Researchers made the point that instead of requesting the user to provide the weights for the skills, these weights can be naturally derived based on the level in the hierarchy, if the taxonomy is constructed in such a way that each level of a concept matches a level of competence. Their

DOI: 10.12948/issn14531305/23.4.2019.05

proposal is still in an early stage, planning to define first a methodology to design the levelized taxonomies.

In [8] is presented a decision support tool, named PROSPECT that helps to shortlist the qualified candidates for a job posting. The architecture of PROSPECT comprises three main components: Batch Processor, Query Processor and Resume Matcher. The batch processor is in fact a nightly job that converts the resumes received over the day in plain text, extracts the information from them (education, work experience and skills together with years of experience for each) and saves it in a DB2 database, detecting also the duplicates. The query processor receives as input the search criteria introduced by the recruiter, parses it and translates it into the suitable format to apply it on the DB2 database. The results coming from the query processor are sent to the resume matcher component that ranks the candidates based on their similarity with the job description (TF-IDF scoring model in Lucene is employed). Pilot tests performed on the tool revealed a roughly sped up in the screening process up to 20 times, compared to the manual approach.

On the other hand, the authors of [9] proposed a reciprocal recommendation algorithm for the recruitment field, claiming that both parties of the process (job-seeker and the employer) should benefit. The paper distinguished between explicit preferences – consisting in what it is mentioned in the candidates' profile and job offer, and implicit preference – derived based on the previous interaction history. Their model implemented next main steps: extracting the users' preference from the resumes and job ad (implicit and explicit preference), calculate the comprehensive similarity between the candidate's profile and the job description and, finally, generate a reciprocal top-n recommendation. Evaluation of the algorithm showed improved accuracy of the recommendation, compared to other existing reciprocal algorithms.

3. The Prototype for an Automatic Resume Screening System

Unlike traditional recommendation systems, such as recommending movies, books or products to the users, job recommendation problem is slightly different. The main difference is that for a job posting we usually want to find few candidates to proceed with the interview phase, whereas the same movie, book, product can be suggested to many users [10]. When a candidate is selected for a particular job, then he becomes ineligible for other job postings.

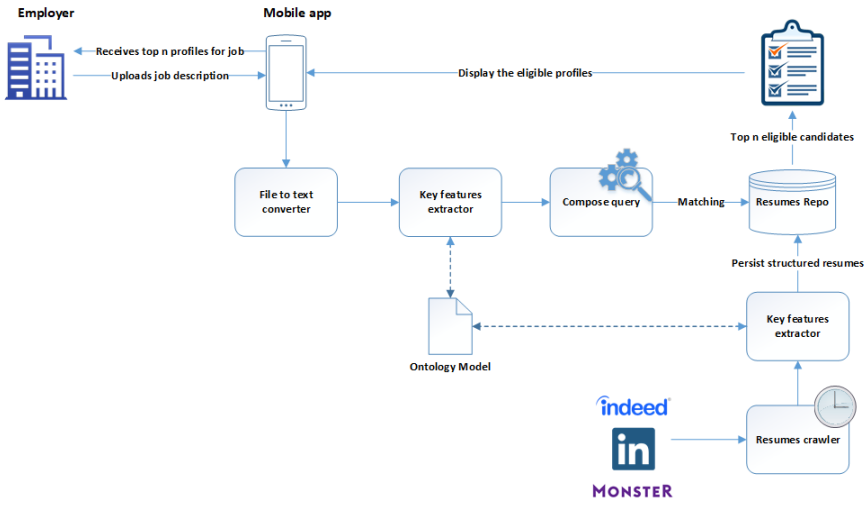

Fig. 1. The architecture of the proposed resume screening system

In order to help recruiters, reach candidates faster, and gain a competitive advantage, we designed a system meant to automate the screening phase of the human resources selection process. Its goal is to provide a valuable support in the first stage of the selection by

DOI: 10.12948/issn14531305/23.4.2019.05

decreasing the number of errors that can occur when thousands of CVs are manually filtered to enter in a detailed analysis or an interview. The architecture of the resume screening system is presented in Figure 1.

The screening system is working in the below described manner:

1) We update on a daily basis a resume repository, extracting data from professional websites (like Indeed, LinkedIn, Monster etc.). The goal is to have an up-to-date pool with candidates, from where we can pick the best ranked for each available job. The repository should be a triple-store as it is meant to store semantic data represented as RDF. Based on the comparative analysis performed in [11] we can choose one of the following considered top triple-stores: AllegroGraph RDF Store, GraphDB (former OWLIM), MarkLogic, Mulgara, Profium Sense, RDF4 (former Sesame), Stardog, Apache Jena–TDB or Oracle Database 12c.
2) Resumes are collected using a crawler that will scan the professional websites in searching of candidates' profiles from the IT market.
3) The information presented in the resumes is extracted and transformed using the key feature extractor component. This component is based on the ontology model (as detailed in Section 4) and creates a RDF from the resume, to be stored in the repository.
4) Companies can upload a job description in the mobile application, in any desired format (.pdf, .txt etc.).
5) Further, the job offer is passed through a text converter that will extract the content from the uploaded file.
6) Then, the job posted by the employers is correspondingly represented as a RDF using the same ontology model. The key feature extractor is responsible to do mapping from the received job info to the RDF representation.
7) A matching query is computed between the resumes in the repository and the job offer posted by the company, to determine the overall similarity and pick the top best candidates for the job.
8) The list with the best ranked candidates whose profiles are matching the requirements is sent to the mobile application that will display to the recruiter the results (link to candidates' profiles in one of the professional job sites).

In can be noticed that the designed prototype is not providing a bi-directional matching, rather it is focused merely on helping the companies to quickly reach the candidates. The reason behind this choice is the fact that in the IT field, due to the significant increase in the demand compared to the offer, the companies are the ones that are hunting down the candidates. Nowadays, the IT professionals are just updating their profiles in the professional networks and are passive seekers. They expect to be contacted for opportunities suitable for their experience, rather than actively searching for a new job.

4 Building the eRecruitment Ontology for Computer Science Field

In order to link the job ads together with the job seekers' profiles we need to ensure that they have a uniform representation, to support the matching. To achieve this goal we designed an ontology that encloses the main aspects when talking about the profile of an IT specialist or about a job description targeting an IT professional.

Ontologies allow users to organize the information in taxonomies of concepts, together with their attributes, and to describe the relationship between these concepts [6]. The authors of [12] highlight two main reasons for using ontologies:

- to share common understanding of the structure of information among people and/or software agents.
- to enable reuse of domain knowledge after it exists.

In the creation of our ontology we started from the model proposed in [13] and extended it, especially focusing on the skills, as these are the key abilities when searching for the ideal candidate. The visual representation of the proposed ontology is depicted in Figure 2.

DOI: 10.12948/issn14531305/23.4.2019.05

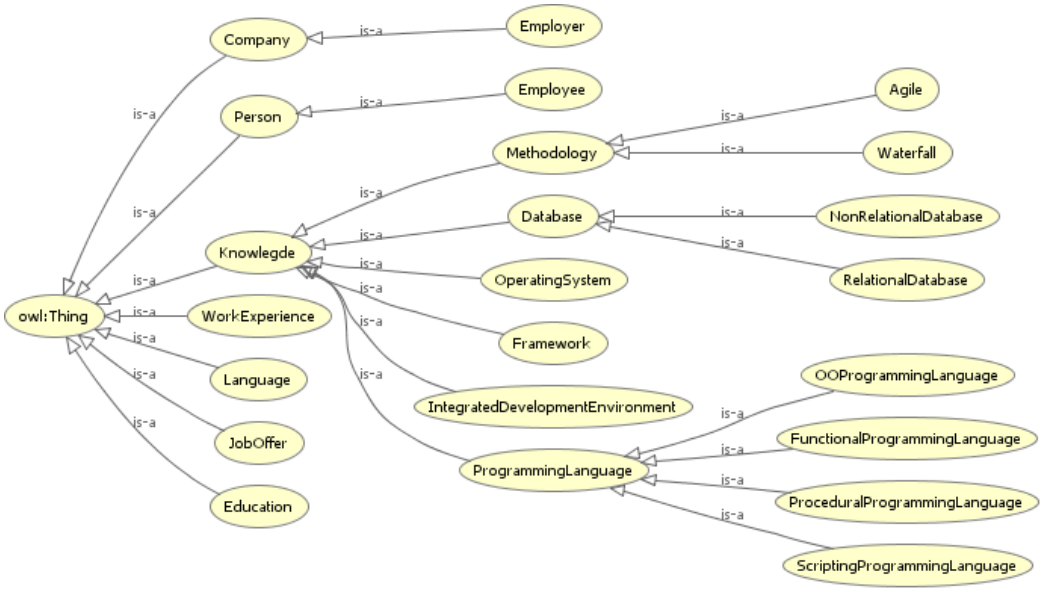

Fig. 2. The classes' hierarchy of the proposed ontology (displayed in OWLViz)

The ontology is composed of the following entities: *Person* with subclass *Employee*, *Company* with subclass *Employer*, *Education*, *Language*, *WorkExperience*, *Knowledge* and *JobOffer*. Each of these entities has its own properties (datatype and object properties). Further we will detail the Knowledge entity, that consists of technical skills or specific competencies a candidate has or a particular job offer requires.

The datatype properties of Knowledge are: name and experience (expressed in years). We divided the knowledge in the following categories: *ProgrammingLanguage*, *IntegratedDevelopmentEnvironment*, *OperatingSystem*, *Framework*, *Database* and *Methodology*. For programming languages we decided to define the next hierarchy: scripting (like Bash, Perl), procedural (like C, Pascal), object oriented (like Java, Kotlin, C++, C#, ObjectiveC etc.), and functional (like Scala). For methodology we distinguish between waterfall and agile. Databases are divided into relational and non-relational. For each of these classes we defined individuals, in order to enclose all the key terms that model the recruitment in the computer science field.

All the entities and their properties, presented as a part of the designed ontology, are used to represent the data about the candidates, the employers and job offers in a RDF format, in order to enable searching based on the semantics of their underlying content.

5. Extracting Skills from the Candidate's Resume

5.1 Technology stack used: GATE, ANNIE, JAPE and Apache Tika

GATE – *General Architecture for Text Engineering* [14], represents an infrastructure that provides necessary tools for developing and delivering of software components that process human language. GATE was developed using Java programming language, starting from 1995, by a group of researchers from the Department of Computer Science of Sheffield University in UK.

GATE excels in text analysis, regardless of the text format and size. From all similar systems, GATE has by far the biggest and diverse users' community across the globe. GATE is open-source and distributed under LGPL licence (Lesser General Public License).

General understanding regarding GATE can refer to one of the following:

- *GATE Developer* – an integrated development environment for human language processing components, together with an information extraction system and a comprehensive set of plugins

DOI: 10.12948/issn14531305/23.4.2019.05

- *GATE Cloud* – a cloud computing solution for hosting large scale text processing applications
- *GATE Teamware* – a web application for collaborative annotations, which enables users to develop projects that create annotations for a collection of documents, providing also facilities even for not trained users that can perform the activity directly from a web browser
- *GATE Mimir* – a search repository used for text indexing and retrieval, annotations, ontologies or semantic meta-data
- *GATE Embedded* – a framework, an optimised library in order to be included into applications, thus giving users access to all services provided by GATE Developer
- *an architecture* – a high level overview about how to build software for language processing
- *a process* – in order to create robust and maintainable services.

The main objective of GATE is to annotate documents with semantics. This process relies on a set of concepts, defined below:

- the document – that will be annotated
- corpus – is a collection of documents, grouped together in order to execute the same process across all of them
- the annotations – created over the document
- annotation types – like "People", "Location"
- annotation sets – containing groups of annotations
- processing resources – used for documents manipulation and annotations creation
- applications – comprising a flow of processing resources, which can be applied over a document or corpus.

GATE can process documents in various formats: TXT, HTML, XML, PDF, Word, etc. After processing these documents can be exported in different formats or stored in a data warehouse for further processing. There are two types of data warehouses: serial (which stores data directly into a directory) or Lucene repositories that allow searching applying Lucene indexing algorithm.

GATE currently supports text analysis in the following languages: English, Spanish, Chinese, Arabic, Bulgarian, French, German, Hindi, Italian, Romanian, Cebuano, Russian and Danish.

GATE is currently used for numerous text processing applications, in areas such as: voice of the customer (sentiment analysis from written reviews or verbal expressed feedback), cancer research, pharmaceutical research, decision support systems, recruitment, exploitation of data present on the web, information extraction, semantic annotations, etc. Among the clients that use GATE we enumerate: The UK National Archives, well known television company – BBC, Innovantage – the global provider of data analysis for the labour market, etc.

Advantages of a solution for text analysis, developed using GATE, are[1]: comprehensiveness, scalability, openness, extensibility and reusability, transparency, robustness and sustainable efficiency.

ANNIE – a *Nearly-New Information Extraction System*

GATE is delivered together with an information extraction system, called ANNIE. ANNIE is a collection of predefined algorithms and is used to create RDF or OWL representations from unstructured content, through the medium of semantic annotations. It is referred to as a nearly new system because it was built upon a pre-existing one, known as LaSIE. The latter was rebuilt and also renamed, preferring this time a human's name as against a dog name, as the team itself points out: "we decided that people are better than dogs at IE" [15].

[1] https://gate.ac.uk/biz/usps.html

DOI: 10.12948/issn14531305/23.4.2019.05

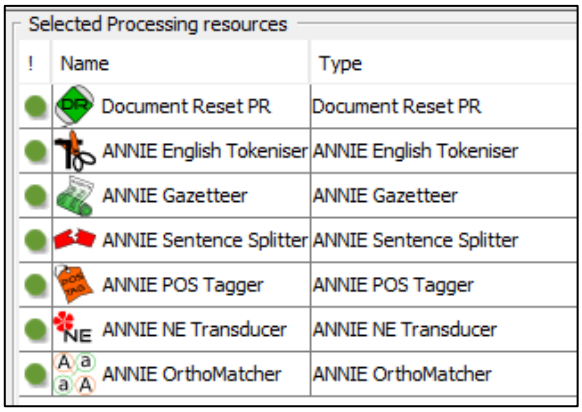

Fig. 3. Processing resources that are part of ANNIE

ANNIE is composed of a set of processing resources that build together a pipeline. The result of one resource is considered input for the next resource in the chain. These resources are represented in Fig. 3. and are detailed below:

1. Document Reset – clears pre-existent annotations, if the document was previously annotated. This step is always applied first in order to avoid duplicated annotations if the same application is executed multiple times on the same document. It can be configured to keep certain annotations sets or to remove only selected types.
2. Tokenizer – it divides the text into small parts, such as numbers, punctuation marks or words of different types. It distinguish between words starting with capital letter or lowercase and also retains the length of each word.
3. Gazetteer – aims to identify entities in the text based on predefined lists. These lists are text files with one record per line, each list representing a set of names: for example, a list for cities, a list for organizations, a list for the days of the week, and so on. All these lists are accessed through an index file called lists.def, and for each word that is found in the text, contained in one of these lists, an annotation of type Lookup is created.
4. Sentence Splitter – defines the beginning and end of each sentence based on the punctuation marks identified by the Tokenizer. To achieve this, a Gazetteer list with abbreviations and a set of rules are applied, which make the difference between punctuation marks that determine the end of a sentence and those that can appear inside.
5. POS Tagger – is a Java implementation of the Brill method, adapted, to associate a tag to each word that characterizes it as part of speech in one of the following categories: adjective, preposition, noun, verb, article, pronoun, adverb, interjection, conjunction etc. The process is also known as grammatical labelling. It uses an implicit lexicon and set of rules and has been trained on a comprehensive set of documents in the Wall Street Journal. Following the execution, a feature called category is added to each annotation produced by Tokenizer.
6. NE Transducer – uses the JAPE language and contains rules based on the annotations created in previous steps to build new annotations. Its role is to eliminate the ambiguity, allowing the combination of several annotations, for example: a date is composed of a day annotation + a month annotation + a number. More details about the JAPE language will be presented in the next subsection.
7. OrthoMatcher – this module works with entities classified as unknown. Different expressions may refer to the same entity, for example DB or Deutsche Bank. The objective of the module is to identify relationships between previously discovered entities and their variants, classified as unknown, and to perform co-referrals. In other words, it will not associate an annotation with an entity that already has one, but starting from the latter it can modify

DOI: 10.12948/issn14531305/23.4.2019.05

the unknown tags with an annotation already created in the previous stages. This component is particularly useful in the case of abbreviations or to identify a family name as being the same as the full name.

After all the annotations were added on a given document, we need to establish their correctness. In the information extracting field the created annotations are classified in one of the next four categories:

- **correct** – properly annotated words, for example Bucharest as a location
- **missing** – words that are not annotated although they should, for example Irina Enăchescu is not marked as a person
- **false positives** – words that are wrongly annotated, for example Irina Enăchescu as a location
- **partially correct** – when the type of the chosen annotation is correct, but the selected entity is not complete, for example only annotating Irina as a person, or more than it should has been selected.

In order to automate the evaluation process, GATE includes the AnnotationDiff instrument that performs a graphical comparison between two sets of annotations. At one point in time, one or two documents and a single type of annotation are compared. The purpose is to make a comparison either between a document annotated via GATE and one manually annotated, or to compare the same annotated document in different versions of the application or through different applications.

Following the comparison, the number of records in each of the four categories is presented to the user and, at the same time, three performance measurement indicators are calculated: precision (accuracy), recall (completeness) and F measure - harmonic mean between the first two indicators.

The precision indicator answers the question: how many of the annotations defined through the application are correct, and is calculated according to the following formula:

$$precision = \frac{number\ of\ correct\ annotations}{number\ of\ correct\ annotations + number\ of\ false\ positives\ annotations}$$

The recall indicator provides an answer for the next questions: how many annotations that should have been created were identified through the application, and is determined by the following formula:

$$recall = \frac{number\ of\ correct\ annotations}{number\ of\ correct\ annotations + number\ of\ missing\ annotations}$$

Depending on how partially correct annotations are taken into account, the AnnotationDiff tool allows us to calculate the three indicators in three distinct variants:

- medium – for partially correct annotations, only half the score is received. This is the most commonly used method.
- strict – only complete annotations are considered correct.
- tolerant – partially correct annotations are counted as correct, in which case the performance results also appear to be better.

JAPE – *Java Annotation Patterns Engine*

JAPE is a language for patterns recognition, specially developed for GATE that works with the concept of grammar. A grammar defined in JAPE language consists in a set of phases, each of them comprising rules of type pattern-action. A grammar has two parts: left-hand-side and right-hand-side. Left-hand-side (LHS) refers to the pattern and contains operators for regular expressions. A label is usually associated to its result. Right-hand-side points to the action that needs to be taken on the recognized pattern, referred through the label, and that action evolves in the form of an annotation that is created.

GATE provides the possibility to chain multiple JAPE grammars, so that a grammar can rely on the annotations created by a previous

DOI: 10.12948/issn14531305/23.4.2019.05

one, thus allowing the creation of more complex annotations. Next we will present an example of a JAPE grammar and we will detail the elements that are used in order to compose it. Starting with the next two sentences: "Player Gheorghe Hagi had 125 games for the national team" and "Model player Gheorghe Hagi was 65 times captain of the national team", we want to create a rule that will link a player annotation with Gheorghe Hagi. The code of this rule can be checked in Figure 4.

```
1  Phase:  GroupedPatterns
2  Input: Lookup Token
3  Options: control = brill
4
5  Rule: player
6  (
7      {Token.string == "Player"}
8      ({Token})?
9  )
10 :label
11 (
12     {Lookup.majorType == Person}
13     |
14     (
15      {Token.kind == word, Token.category == NNP, Token.orth == upperInitial }
16      {Token.kind == word, Token.category == NNP, Token.orth == upperInitial }
17     )
18 )
19 :sportsman
20 -->
21  :sportsman.Player = {rule = "player"}
```

Fig. 4. JAPE grammar example

The rule is explained as follows:

Line 1: Includes the name of the phase. The phase name is unique and does not necessarily has to be the same as the name of the .jape file in which the grammar resides.

Line 2: Refers to the input annotations that will be used by the rule. In this particular case we have two input sources for the annotations: Lookup (obtained after applying Gazetteer resource from ANNIE) and Token (obtained after applying Tokenizer resource, as part of ANNIE).

Line 3: Defines an option called grammar control that can have one of the following values: appelt, brill, once, first and all. In order to understand how they work we will explain them based on an example. Considering the text "Irina Mihaela Enăchescu" and the pattern {Name}+, based on the selected matching style we will obtain the next results as depicted in Table 1.

Table 1. Matching styles in JAPE (adapted after [16])

Appelt (the longest pattern)	Irina Mihaela Enăchescu
Brill (all the combinations starting from the beginning)	Irina, Irina Mihaela, Irina Mihaela Enăchescu
Once (only the first match, then stop)	Irina
First (first match)	Irina, Mihaela, Enăchescu
All (all the matching patterns)	Irina, Irina Mihaela, Irina Mihaela Enăchescu, Mihaela, Mihaela Enăchescu, Enăchescu

Line 5: Declares the name of the rule.

Line 7: Checks if the word contains the text "Player".

Line 8: Points to the fact that after the text "Player" can be present, but is not mandatory, another word. This behaviour is obtained though the usage of "?" operator. Beside this operator there are also another ones that can

DOI: 10.12948/issn14531305/23.4.2019.05

be used, like: "*" – marking zero or more occurrences or "+" – suggesting one or more occurrences.

Line 10: Establishes a label for the pattern that was used in order to identify the text "Player" which can be followed, but not necessarily, by another word.

Line 12: Uses the annotations gathered through Gazetteer and checks if after applying the predefined lists of words a person is identified.

Line 13: Includes the | (or) operator that links two conditions.

Lines 15 and 16: Introduce a pattern to identify two words, nouns in the singular, each starting with a capital letter. They represent the last name and the first name of a person.

Line 19: Defines a label for the pattern used to recognize a person by his/her name.

Line 20: Splits the left-hand-side of the grammar, placed above the arrow, from the right-hand-side, located below.

Line 21: It uses the label defined to the left of the rule to build the Player annotation, providing information that was created by the "player" rule.

Apache Tika consists in a toolkit for extracting content and metadata from different document types, like HTML, XML, documents created using Microsoft Office suite (Word, Excel, PowerPoint), PDF, RTF and even multimedia files, like JPEG, MP4 etc. [17]. All these document types are handled using a single common interface, called Parser. This feature turns Tika into a powerful and versatile instrument for text analysis. Apache Tika proves itself useful for content analysis, translations, indexing for search engines etc. Development of this instrument started in 2004, under Apache Nutch project, so that 6 years later, in 2010, to become one of the top projects of Apache. Apache Tika can automatically determine the type of a document and also the language it was written in. When determining the document's type, not only the extension is used, but also the extracted metadata, together with the analysis of the octets from the beginning of the file, known as magic octets.

Apache Tika is reusing existing libraries that proven good results for content extraction from various file types, like ApachePOI (used for Microsoft Office documents) or PDFBox (for documents in pdf format). This instrument is permanently updated, during one year being released multiple improved versions.

5.2 Implementation and Results

First step in doing an automatic filtering of the candidate's CVs for a job opening in the IT field is to extract the knowledge they possess that is encompassed in their resumes. After this stage the selected information will be used to create a structured RDF document, which in turn will be stored into a triple-store that will be queried for each job offer. We have to keep in mind that, most of the times, the CVs can be found in different formats (.doc, .docx, .pdf etc.), unstructured, and they are written in human language, intelligible to people, but harder to interpret by computers.

In order to implement the knowledge extraction from the resumes we used GATE, described earlier. We installed GATE Developer, version 8.4.1[2], available since 9 June 2017. To configure GATE so as to identify the knowledge present in a CV, a series of steps were required, detailed below:

1. a KnowledgeSchema.xml file was created and saved in the next location: GATE_Developer_8.4.1\plugins\ANNIE\resources\schema
2. the previous created file was added as a new entry in: GATE_Developer_8.4.1\plugins\ANNIE\resources\schema\ANNIE-Schemas.xml
3. a knowledge.lst file was built and saved in GATE_Developer_8.4.1\plugins\ANNIE\resources\gazetteer. Here is defined a list of words that represent knowledge for a candidate profile in the IT field, such as programming languages, operating systems, databases, development environments, etc.

[2] https://gate.ac.uk/download/

DOI: 10.12948/issn14531305/23.4.2019.05

4. the file created in step 3 was added in: GATE_Developer_8.4.1\plugins\ANNIE\resources\gazetteer\lists.def
5. several rules were created in .jape files and stored in: GATE_Developer_8.4.1\plugins\ANNIE\resources\NE
6. all the rules filenames created at step 5 were added in: GATE_Developer_8.4.1\plugins\ANNIE\resources\NE\main.jape

An example of one of the defined rules can be seen in Fig. 5. This rule specifies that if after the search process (identifying the words in the list defined in knowledge.lst) a word was marked as representing a skill, and it is followed by a comma, then the next word after the comma must be marked also as knowlegde, even if it is not in the defined list.

knowledge_category_nested1.jape

```
/*
*  knowledge_category_nested1.jape
*  Irina Enachescu, 23 June 2019
*/
Phase:  KnowledgeCategoryNested1
Input: Lookup Token
Options: control = appelt

Rule: KnowledgeCategoryNested1
Priority:40
(
    (
        {Lookup.majorType == knowledge}
        {Token.kind == punctuation, Token.category == ","}
    )
)
:temp
(
    ({Token.kind==word})+
)
:label
-->
 :label.Knowledge = {rule = "KnowledgeCategoryNested1"}
```

Fig. 5. Rule example for knowledge identification in a CV

Once the new annotation (knowledge) was defined, the list of words to be marked as part of this category and the rules to help identify as many competencies as possible in the content of a CV were created, the next step was to validate these steps.

In the *Language Resources* section we added a new GATE document, in which a CV was uploaded. Then we created, in the same section, a GATE corpus, a collection of documents, in which the newly defined document was included. Subsequently, from the top left menu we loaded the ANNIE system, with default parameter settings. Finally we selected the document collection and executed ANNIE over them. In Figure 6. is depicted the application after all the previous described stages were correctly executed, and in Figure 7 it is shown how the CV looks after it was annotated with the identified competencies, either from the defined list or based on the implemented rules.

DOI: 10.12948/issn14531305/23.4.2019.05

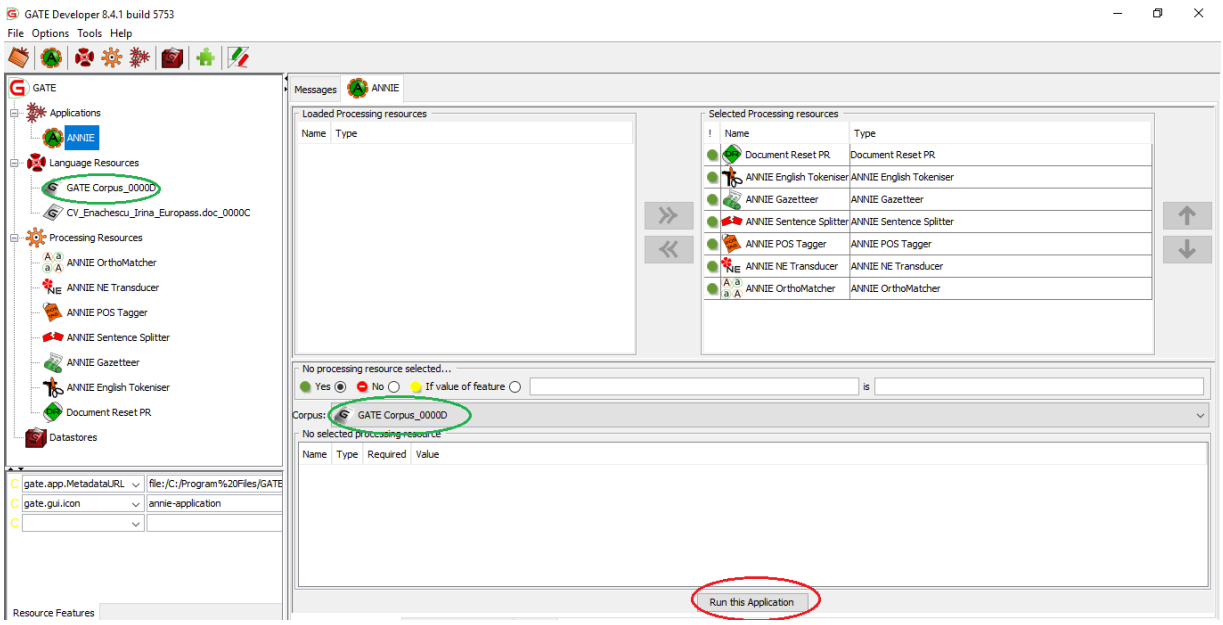

Fig. 6. Executing ANNIE information extracting system over a CV

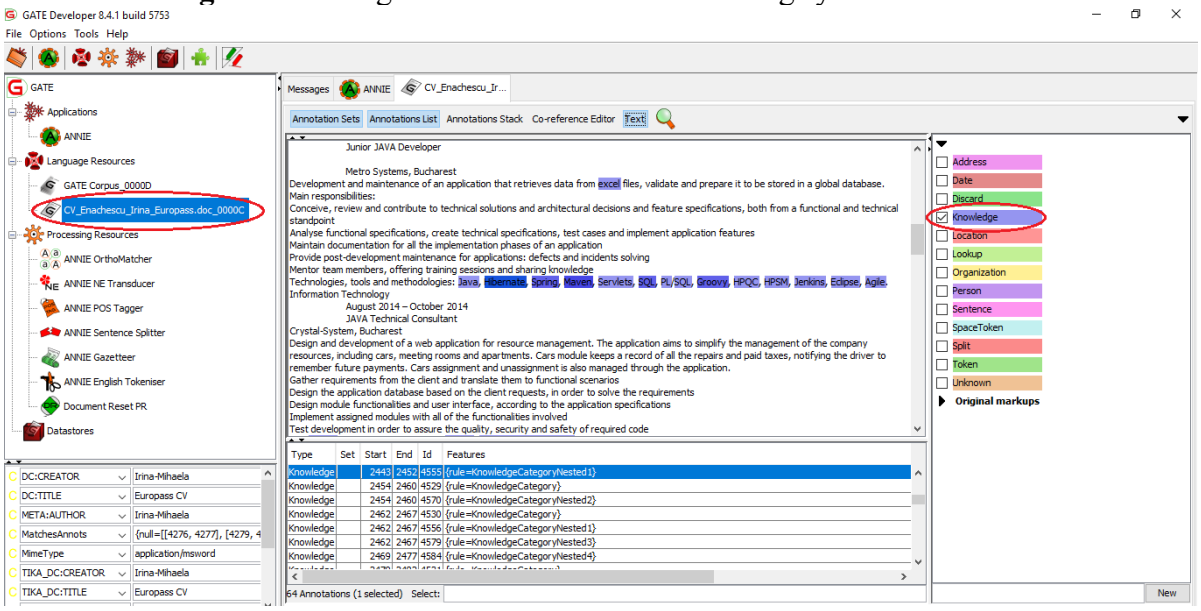

Fig. 7. Annotated CV content after applying ANNIE

After we proved the CV was correctly annotated using GATE Developer, we continued next with implementing a Java application, which having as input data the path to a directory where multiple CVs are stored, in various formats (.doc, .docx, .pdf), generates for each CV a list with all the extracted competencies. Source code of the application can be examined in Listing 1.

Listing 1 – Source code for extracting the competencies from a collection of CVs

```
public class ResumeKnowledgeExtractorApp {
    private static final Logger LOGGER = LoggerFactory.getLogger(ResumeKnowledg-
eExtractorApp.class);
    private static final String RESUMES_FOLDER = "E:\\CVs";
    public static void main(String[] args) {
        LOGGER.info("Starting extracting the knowledge/skills from resumes saved
in {} folder ...", RESUMES_FOLDER);
        File resumesFolder = new File(RESUMES_FOLDER);
        if (resumesFolder.isDirectory() == true) {
            List<File> resumes = Arrays.asList(resumesFolder.listFiles());
```

DOI: 10.12948/issn14531305/23.4.2019.05

```
            try {
                ResumeGate resumeGate = new ResumeGate();
                resumeGate.init();
                resumeGate.initAnnie();
                Corpus corpus = Factory.newCorpus("Resumes_Corpus");
                //add all resumes in corpus
                resumes.stream().forEach(r -> {
                    try {
                        Document document = Factory.newDocument(new FileCon-
tentExtractor(r.getPath()).extract());
                        document.setName(r.getName());
                        corpus.add(document);
                    } catch (ResourceInstantiationException | IOException |
TikaException | SAXException e) {
                        LOGGER.error("An error occurred", e);
                        throw new RuntimeException(e);
                    }
                });
                resumeGate.setCorpus(corpus);
                resumeGate.execute();
                Iterator iterator = corpus.iterator();
                while (iterator.hasNext()) {
                    Document doc = (Document) iterator.next();
                    Set<String> skills = doc.getAnnotations().get(new
HashSet<>(Arrays.asList("Knowledge")))
                            .stream()
                            .map(annotation -> Utils.stringFor(doc, annotation))
                            .collect(Collectors.toSet());
                    LOGGER.info("For document {} we detected the following
skills: {} ", doc.getName(), skills);
                }
                resumeGate.cleanUp();
            } catch (GateException | IOException e) {
                LOGGER.error("An error occurred", e);
                throw new RuntimeException(e);
            }
        } else {
            LOGGER.warn("Please provide a folder with CVs as input.");
        }
        LOGGER.info("Knowledge extracting completed.");
    }
}
```

Two main classes enclose the application logic: FileContentExtractor and ResumeGate. First of them, receives at defining point, in the constructor, the path to a file, and has an execute() method that is responsible to return the content of the file as a String. For the purpose of content extraction Apache Tika framework was employed. The ResumeGate class provides the necessary methods to interact with the API provided by GATE. It contains the methods: init() - which initializes in the Java application the embedded version of GATE, initAnnie() - which loads the ANNIE system with implicit parameters, setCorpus() - which associates a collection of documents to the application, execute() – for running the application and cleanUp() - which ensures the release of the resources used, to avoid generating memory leaks.

The result for a CV, as extracted from the console logging, is presented below, in Listing 2:

Listing 2 – Application result after testing it with a CV

```
15:16:57.687 [main] INFO com.ase.irina.enachescu.resume.screener.parser.Re-
sumeKnowledgeExtractorApp - For document CV_Europass.doc we detected the follow-
ing skills: [C#, Maven, C, RxJava, HPSM, SAS, Eclipse, HTML, Groovy, Java 8, Ja-
vaScript, Agile, Apache Camel, Control-M, Java, C++, Hibernate, CSS, Jboss tools
extensions, JSP, SCRUM, Spring, SQL, RMI, excel, Jenkins, jQuery, Git, Spring
Framework, HPQC, PL, Servlets, Intellij IDEA]
```

DOI: 10.12948/issn14531305/23.4.2019.05

The functionality of this application will be exposed through a REST service that receiving as input a list of CVs will determine for each of them what the relevant competencies are.

6 Conclusions and future work

In order to help the HR personnel to reach the candidates in the IT field faster, and gain therefore competitive advantage, we proposed a system meant to automate the CVs screening phase of the candidates' selection process. In order to link the job ads together with the job seekers' profiles we need to ensure that they have a uniform representation, and for this purpose we designed an ontology that encloses the main aspects that need to be considered when doing recruitment in the computer science field. We also proposed the architecture of a system that using this ontology can be developed as a tool to enhance the efficiency of recruiters.

Second part of this paper focuses on the implementation of an application that extracts the competencies from the candidates' CVs. This application represents one of the key modules from the proposed prototype. We introduce the used tools and libraries, like GATE – an infrastructure that provides necessary tools for developing and delivering of software components that process human language – and Apache Tika – for documents content extraction. In the end, we describe in detail the developed application, written in Java, which having as input data the path to a directory where multiple CVs are stored, in various formats (.doc, .docx, .pdf), generates for each CV a list with all the extracted competencies.

As a next step in this research, we intend to continue the development of the candidates' screening platform and test it using real-world data sets. Directions for future research include also the extension and refinement of the proposed ontology, because the robustness and completeness of the modelled ontology is directly impacting the accuracy of the system that uses it.

A different approach that is it also worth to consider is the new LinkedIn search paradigm, called *Search by Ideal Candidates*, proposed by [18]. This implies to select a small set of ideal candidates (from one to three) for a position, instead of providing a job description. The ideal candidates can be from the ones already working on a similar position in the company. The system then builds a query automatically composed from the key information in the profiles of the input candidates and searches for suitable candidates who are similar to them. This approach is known as item to item recommendation.

Acknowledgment

Parts of this research have been published in the Proceedings of the 17th International Conference on Informatics in Economy, IE 2018 [19].

References

[1] H. Bukvova, "A holistic approach to the analysis of online profiles", Internet Research, vol. 22, no. 3, pp. 340-360, 2012.

[2] S. Pande, "E-recruitment creates order out of chaos at SAT Telecom", Human Resource Management International Digest, vol. 19, no. 3, pp. 21-23, 2011.

[3] S. T. Al-Otaibi and M. Ykhlef, "A survey of job recommender systems", International Journal of the Physical Sciences, vol. 7, no. 29, pp. 5127-5142, 2012.

[4] E. Pariser, "Beware online "filter bubbles"", Ted.com, 2011. [Online]. Available: https://www.ted.com/talks/eli_pariser_beware_online_filter_bubbles. [Accessed: 02- Jun- 2019].

[5] V. Kumaran and A. Sankar, "Towards an automated system for intelligent screening of candidates for recruitment using ontology mapping (EXPERT)", International Journal of Metadata, Semantics and Ontologies, vol. 8, no. 1, pp. 56-64, 2013.

[6] M. Uddin, T. Duong, K. Oh, J. Jung and G. Jo, "Experts search and rank with social network: An ontology-based approach", International Journal of Software Engineering and Knowledge Engineering, vol.

DOI: 10.12948/issn14531305/23.4.2019.05

23, no. 01, pp. 31-50, 2013.
[7] M. Guedj, "Levelized Taxonomy Approach for the Job Seeking/Recruitment Problem," 2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES), Paris, 2016, pp. 448-451.
[8] A. Singh, C. Rose, K. Visweswariah, V. Chenthamarakshan and N. Kambhatla, "PROSPECT: a system for screening candidates for recruitment", in Proceedings of 19th ACM International Conference on Information and Knowledge Management (CIKM'10), Toronto, Ontario, Canada, 2010, pp. 659-668.
[9] H. Yu, C. Liu, F. Zhang, "Reciprocal Recommendation Algorithm for the Field of Recruitment", Journal of Information & Computational Science, vol. 8, no. 16, pp. 4061-4068, 2011.
[10] K. Kenthapadi, B. Le and G. Venkataraman, "Personalized Job Recommendation System at LinkedIn: Practical Challenges and Lessons Learned", in RecSys '17 Proceedings of the Eleventh ACM Conference on Recommender Systems, Como, Italy, 2017, pp. 346-347.
[11] B. Iancu and T. Georgescu, "Saving Large Semantic Data in Cloud: A Survey of the Main DBaaS Solutions", Informatica Economica, vol. 22, no. 1/2018, pp. 5-16, 2018.
[12] I. Abuhassan and A. AlMashaykhi, "Domain Ontology for Programming Languages", Journal of Computations & Modelling, vol. 2, no. 4, pp. 75-91, 2012.
[13] M. I. Enachescu, "A Prototype for an e-Recruitment Platform using Semantic Web Technologies", Informatica Economica, vol. 20, no. 4/2016, pp. 62-75, 2016.
[14] Cunningham, et al. Developing Language Processing Components with GATE Version 8. University of Sheffield Department of Computer Science. 17 November 2014. [Online]. Available: https://gate.ac.uk/sale/tao/
[15] "Module 2: Introduction to IE and ANNIE", Gate.ac.uk. [Online]. Available: https://gate.ac.uk/sale/talks/gate-course-may10/track-1/module-2-ie/module-2-ie.ppt. [Accessed: 02- Jun- 2019]
[16] "Module 3: Introduction to JAPE", Gate.ac.uk. [Online]. Available: https://gate.ac.uk/sale/talks/gate-course-aug10/track-1/module-3-jape/module-3-jape.pdf. [Accessed: 02- Jun- 2019]
[17] N. Thai, "Content Analysis with Apache Tika | Baeldung", Baeldung, 2019. [Online]. Available: http://www.baeldung.com/apache-tika. [Accessed: 02- Jun- 2019]
[18] V. Ha-Thuc, Y. Xu, S. Pradeep Kanduri, X. Wu, V. Dialani, Y. Yan, A. Gupta and S. Sinha, "Search by Ideal Candidates: Next Generation of Talent Search at LinkedIn", in WWW '16 Companion Proceedings of the 25th International Conference Companion on World Wide Web, Montréal, Québec, Canada, 2016, pp. 195-198.
[19] M.I. Enăchescu, "Screening candidates in IT field using an ontology based approach", in Proceedings of the 17th International Conference on Informatics in Economy (IE 2018), Iași, Romania, 2018, pp. 303-308

Mihaela-Irina ENĂCHESCU has graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 2014. In 2016 she has graduated the Economic Informatics Master program and she currently pursues a PhD research in Economic Informatics at the Bucharest University of Economic Studies. She is working as a Java Software Developer and is also a teaching assistant in the Department of Economic Informatics and Cybernetics. Her research interests are: Ontologies, Semantic Web and Data Mining.

DOI: 10.12948/issn14531305/23.4.2019.05