KESeDa: Knowledge Extraction from Heterogeneous Semi-Structured Data Sources

Martin Seidel
Technische Universität Chemnitz
Chemnitz, Germany
martin.seidel@informatik.tu-chemnitz.de

Michael Krug
Technische Universität Chemnitz
Chemnitz, Germany
michael.krug@informatik.tu-chemnitz.de

Frank Burian
Technische Universität Chemnitz
Chemnitz, Germany
frank.burian@informatik.tu-chemnitz.de

Martin Gaedke
Technische Universität Chemnitz
Chemnitz, Germany
martin.gaedke@informatik.tu-chemnitz.de

ABSTRACT

A large part of the free knowledge existing on the Web is available as heterogeneous, semi-structured data, which is only weakly interlinked and in general does not include any semantic classification. Due to the enormous amount of information the necessary preparation of this data for integrating it in the Web of Data requires automated processes. The extraction of knowledge from structured as well as unstructured data has already been the topic of research. But especially for the semi-structured data format JSON, which is widely used as a data exchange format e.g., in social networks, extraction solutions are missing. Based on the findings we made by analyzing existing extraction methods, we present our KESeDa approach for extracting knowledge from heterogeneous, semi-structured data sources. We show how knowledge can be extracted by describing different analysis and processing steps. With the resulting semantically enriched data the potential of Linked Data can be utilized.

CCS Concepts

•**Information systems** → *Resource Description Framework (RDF);* **Data extraction and integration;**

Keywords

knowledge extraction, semantic enrichment, semi-structured data, semantic web, web of data

1. INTRODUCTION

Data on its own contains only a weak information value. However, if we interconnect it with other data and structural information, specific relations can be identified and knowledge can be inferred. A large amount of knowledge existing on the Web is available as weakly interlinked, heterogeneous, semi-structured data, which in general does not contain any semantic classification. Furthermore, knowledge is partially created by users themselves, especially in the context of social networks. This user-specific data and the interconnections are, depending on the privacy setting and the linkage to other user data, open accessible for everybody.

As a consequence, public domain data and its contained knowledge is more and more in the center of attention of enterprises and governments (E-Government). Enterprises can also benefit of this knowledge by evaluating specific customer data, e.g., by generating appropriate recommendations for online shop visitors. Amazon is an excellent example using this strategy already for years. The use of plain customer data is usually not sufficient to accomplish good recommendations. Moreover, additional metadata and knowledge about the products themselves as well as their relations are necessary to recommend similar or comparable products.

Fortunately, a large amount of those kinds of data is already available publicly on the World Wide Web. Approximately 1.64 billion people actively use the social network Facebook[1], 310 million people the short message service Twitter[2], and 4.75 billion content elements are shared on Facebook worldwide every day[3]. These content elements include pictures, videos, comments and status updates. In comparison to text-based content, the percentage of shared multimedia content is still relatively low. Therefore, textual data represents a good basis to extract knowledge. Since the amount of newly created data per day constantly increases, the preparation and utilization requires automated processes. This incorporates the extraction as well as the reconditioning to gain high quality data sets.

The mentioned available data does exist in different for-

SEMANTiCS 2016, September 12-15, 2016, Leipzig,Germany
© 2016 ACM. ISBN 978-1-4503-4752-5/16/09...$15.00
DOI: http://dx.doi.org/10.1145/2993318.2993335

[1]http://de.statista.com/statistik/daten/studie/37545/umfrage/anzahl-der-aktiven-nutzer-von-facebook/
[2]http://de.statista.com/statistik/daten/studie/232401/umfrage/monatlich-aktive-nutzer-von-twitter-weltweit-zeitreihe/
[3]http://www.futurebiz.de/artikel/facebook-statistiken-475-mrd-inhalte-werden-taeglich-auf-facebook-geteilt/

mats. One of them and probably the most common one is semi-structured data. In semi-structured data there are no defined types and the structure is only defined by the purpose of the data. Based on this characteristic, semi-structured data offers good extensibility and flexibility for data exchange. Within our work, when speaking about semi-structured data, we refer to the formats HTML, XML and JSON.

The intended use determines the amount of semi-structured metadata, e.g., users sharing content they like with their friends on social networks. Friends lists provide information about social structures and the environment of users. Data from social networks is often publicly accessible through service-specific APIs using a semi-structured representation. This enables the utilization of this data to extract knowledge.

The extracted knowledge can be stored in different formats, preferably using the RDF format. RDF offers to save knowledge semantically and enables the interlinking of data between different knowledge bases. This is very special feature, because thereby it is possible to map, e.g., a schema.org/person to a foaf/person using ontology languages like OWL so that a search can be performed over multiple knowledge bases. SPARQL can be used to perform such a query on the basis of existing data, e.g., finding all users on Facebook that became friend with person X and did not post a status message this summer stating they ate ice cream. A prerequisite for this query is that the required information was extracted and that the given information is correct. Common search engines like Google or Bing are still limited in executing such a query because their algorithms do not yet semantically focus on the interrelation between the contained object concepts. To obtain comprehensive statements a broad knowledge base is required. Thus, many extracted, qualitatively high data sets are needed. Characteristics like accuracy, coherence and correctness are important.

To address the mentioned need for automatic knowledge extraction solutions, we propose KESeDa (**K**nowledge **E**xtraction from heterogeneous, **Se**mi-structured **Da**ta Sources) as an approach for extracting knowledge from heterogeneous, semi-structured data sources.

The rest of the paper is structured as follows: In the second chapter, we give an overview of existing solutions to extract knowledge from semi-structured data sources, especially based on HTML and XML. Based on the analysis results, we present our KESeDa approach to perform knowledge extraction on data represented as JSON in the third chapter. The main contribution is an algorithm that we divide into multiple components of different responsibility in order to increase its re-usability. In the fourth chapter we evaluate our approach. The paper is concluded with a summary and outlook on future work.

2. RELATED WORK

For the extraction of knowledge from semi-structured data sources a variety of algorithms and software tools already exist. Most of them can be clustered into groups based on similar principles or implementing the same logic. Often these solutions only vary in their field of application. Alberto H. F. Laender created such a classification schema for data extraction solutions using six different clusters within the paper ”A brief survey of web data extraction tools” [21]. We will reuse this approach, because we determined that all current tools and software solutions can be classified in these groups even nowadays.

Wrappers are a common approach to extract data from semi-structured data sources [21]. A wrapper is nominally some kind of cover, closure or envelope. Kushmerick defines a ”wrapper in data mining (as) a program that extracts content of a particular information source and translates it into a relational form” [19]. The basic idea is to manually develop such a wrapper for every website (or part of a website) where data should be extracted from. The amount and the quality of the data extracted is significantly dependent on the accuracy and the correctness of the constructed wrapper. The main advantage of this strategy is that such a wrapper can be applied to any website and will deliver good results [4]. A significant challenge in the application of wrappers is the separation of relevant and irrelevant data, like markup or inline code [21]. Unfortunately, the creation of these wrapper is a very time-consuming task because they have to be defined manually and individually for each specific task. They are bound to the structure of the website and therefore will certainly not work (or only with limitations) if the underlying DOM tree was changed [19]. It would be necessary to adjust the wrapper according to the new site structure in order to maintain the information extraction. Thus, the next reasonable step is to automatically generate such a wrapper by using different techniques. This method is referred to as Wrapper Induction [19].

The group of *wrapper induction tools* is based on the approach of generating extraction rules based on separator symbols. The generation is done automatically based on several training sets [21]. Those training sets are produced with a process called ”labeling” [4], where a human interaction is required to annotate the valuable information in the underlying data sources. Based on this information, an algorithm attempts to create a wrapper [4]. In contrast to NLP-based approaches, wrapper induction tools extract data based on its structural characteristics instead of using linguistic limitations. Hence, these kind of wrappers utilize the formatting of the data in order to identify the relevant information. Exemplary tools that implement this approach are WIEN [20], SoftMealy [16], Stalker [26], OMI [6], d2c [30], REX [8], Extraction Heuristics [4] or FINDER [2].

One of the first solutions for generating wrappers was to develop specific languages [21]. Such a language contains the usage of declarative grammars, procedural programming languages or declarative query languages [4]. Several languages were designed in order to assist the user in creating wrappers [10]. However, the user has to build the wrapper on his own. Examples implementing this approach are Minerva [11], TSIMMIS [15], Web-OQL [3], Florid [23] and Jedi [17].

Natural language processing (NLP) is an approach to utilize the characteristics of natural language to learn rules for the extraction of relevant data elements [21]. Examples of such techniques are filters, ”part-of-speech tagging” or ”lexical semantic tagging”. They are used to deduce relations between sentences and containing elements to derive extraction rules [21]. These rules are also based on syntactical and semantical conditions that help to identify relevant information in a certain document [21]. The NLP approach attempts to extract data based on natural speech processing, so it is especially suitable for text within semi-structured data sets and less applicable for the direct analysis of semi-structured documents like HTML pages [21].

”Part-of-speech tagging” denotes rule-based techniques that are subsequently processed on a plain text. The advantages are that these rules can easily be created manually and that already a few rules are enough to provide results. However, these rules are often language-specific and require high effort for achieving really good results [27].

"Lexical semantic tagging" is another approach, which assigns every word in a text to its suitable lexical meaning by using different methods and techniques [31]. It also incorporates the principle of compositionality, which states that the meaning of a complex expression is determined by the meaning of its parts (Frege's principle) [13]. Tools implementing this approach are RAPIER [25], SRV [14], WHISK [29] and PIKES [9].

HTML-based approaches use the existing structural characteristics of HTML documents [4], which are defined by the W3C standard[4]. The HTML document is parsed into a tree model based on the original DOM structure before starting the extraction process [21]. Extraction rules are created (semi-)automatically and then applied on the document [4]. Examples for tools implementing this approach are W4F [28], XWRAP [22], RoadRunner [12], and Lixto [5].

Model-based tools try to extract data on the basis of a given model. The starting point is a set of target models. An algorithm tries to structurally recognize these models on a web page, e.g., by iterating through a list and comparing it with a specific structure. Known examples implementing this approach are NoDoSE [1] and DEByE [21].

The previously presented approaches are all based on the structure of the source document to generate specific rules or patterns in order to extract relevant data. Ontology-based tools however use another method. Their extraction processes are directly referred to the data. Tools implementing the ontology-based approach are for example DAMASK [24] and "An Ontology-based Retrieval System Using Semantic Indexing" [18].

2.1 Summary

The analysis has shown that many approaches for extracting semi-structured data suffer from limitations regarding the support for different formats. Many approaches only support the HTML and XML format. An exception are NLP-based methods, which do not support semi-structured data but unstructured text. Ontology-based formats on the other hand facilitate semi-structured data and texts to the full extend. The support for JSON is in general not common. Heterogeneous data sources are supported by all described approaches. Some methods have to be adjusted or trained to new data sources in advance. However, no technique is fixed to a particular data source. With respect to the level of automation huge discrepancies exist between the different approaches. Whereas the wrapper induction approach can work fully automated, NLP, HTML- and model-based tools perform their work at least semi-automated. Languages for wrapper generation and ontology-based methods have to be applied manually. Except the NLP approach, which requires a high learning effort to generate rules, all other solutions do not need an extensive training. Based on these findings, we will now present our knowledge extraction approach for semi-structured data.

3. THE KESeDa APPROACH

The examination and evaluation of existing approaches has shown that the extraction of knowledge from JSON data is not supported sufficiently so far. Thus, we propose the KESeDa (**K**nowledge **E**xtraction from heterogeneous, **Se**mi-structured **Da**ta Sources) approach. Our general approach is separated into multiple processing steps. The overall idea is to firstly determine the data format of the given source data followed by distinct analysis and processing steps. Since the extraction from HTML and XML data sources is already adequately investigated, we propose to handle the extraction process of those formats by utilizing already existing and proven solutions. If the JSON format is detected, we will apply our own algorithm. A conceptual component diagram of the approach is illustrated in Figure 1.

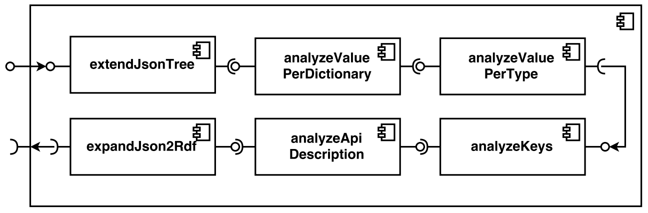


Figure 1: Conceptual component diagram of the KESeDa Approach

The first step covers the preparation of the source file for later annotations. Therefore, all values contained within the JSON object are encapsulated in a separate object. This object additionally contains an array structure as a placeholder in order to store all identified properties that are may be assigned to predicates during the following processes. Each potential predicate has attached a relevance value and a reference count. We can use this count to infer how often the specific property was referenced correctly later on. It is also possible that the same predicate has multiple references. For example a "surename" could be a "sure-name", "sure.name" or "sureName". In this case, we store all those references inside the prepared structure, which will be later resolved to the appropriate predicate.

Afterwards, the values are analyzed by matching them against a set of dictionaries. The gathered results are stored in the aforementioned placeholder array. It is also possible to combine multiple dictionaries to map composed predicates like a name, which consist of a first name and a last name.

Another analysis step examines the property values with focus on their data type and format.

Following, the keys of the JSON object are analyzed. If the name of a key exactly matches a predicate, it will be stored in the array. If the key can not yet be assigned to a known predicate, synonyms for the key are looked up in a dictionary and rated regarding a possible mapping. If the source data was provided using an API, a subsequent processing step attempts to extract metadata based on the API documentation.

A final component transforms the annotated JSON source object into a JSON-LD representation by selecting a suitable RDF predicate for each property. In the following, we attempt to find an appropriate RDF class for each object based on the set of its predicates, e.g., a class Person that is represented by its name and its birthday. As a final post-processing step, a conversion and validation component checks the result for syntactic correctness and transforms it to the specified target RDF format. In the following subsections, we describe

[4]W3C HTML - http://www.w3.org/html/

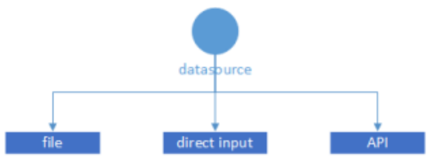


Figure 2: Distinction of different data sources

the single analysis and processing steps in more detail.

3.1 Specifying Configuration Options

At first, we provide the user with various configuration options. We enable to explicitly specify the input format like HTML, XML, JSON or Unknown. Furthermore, the output file name and format can be chosen. The format can be e.g., RDF/XML, JSON-LD, N-Triples, Turtle Terse Triple Language or Notation 3. Additionally, there is an option that will enable the user to get a quick preview of the assigned predicates before generating the output file in order to correct mistakes manually. Those corrections are used to rate the automatically applied rules. A second feature offers to enable the link processing. Which means that the algorithm will process and follow links stated within the source document up to a specific depth and will also process the retrieved data. This option is interesting e.g., for social media data.

3.2 Retrieving Source Data

As the entry point of our approach we define the retrieval of the source data. We offer the user to upload data as a single file or enter it in a web form. Another alternative is to retrieve data using a specified API endpoint. In the latter case, both HTTP methods GET and POST in combination with parameters are supported since some APIs require authentication. Figure 2 illustrates these options schematically. In our diagrams the circle represents the start of the algorithm, the arrows visualize the control flow, the rectangles mark activities for specific cases that are determined by analysis steps represented by diamonds.

3.3 Detecting Data Format

As a first analysis step, the format of the source data will be identified. As mentioned before, the user can specify the format in advance. If this information was stated, we only test if the indication matches the actual data. Figure 3 provides a brief description for each examination step. In the case the user uploaded a file, the algorithm runs severals checks. The first option is to check the file extension. Additionally, internal metadata can be used to detect the format. This includes the file header or a magic number, which e.g., can also be an included document type definition or XML identifier. Furthermore, the format is evaluated by checking whether the data can be parsed into an object using different parsers. E.g., trying to parse the data into a JSON object. If the data source was given as an API-URI, the HTTP header of the response is analyzed (i.e., checking the Content-Type entry). The combination of all analysis methods is used to identify the most likely format.

3.4 Processing HTML and XML Data

If the analysis has identified the data format as HTML or XML, we base the extraction on existing solutions, since as mentioned in Section 2, extraction methods for HTML

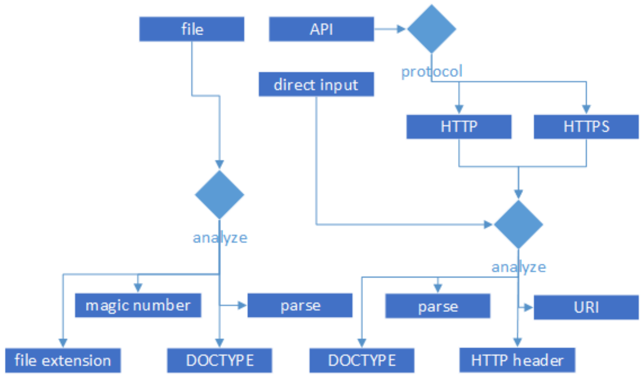


Figure 3: Analysis of the data format

and XML documents were already adequately in the focus of research. We propose to use solutions like the REX Project [7] that is able to extract knowledge from HTML/XML files and generates corresponding RDF/XML files, which are then converted into a target format. In case the source data was provided as JSON, we will apply our own extraction approach as described in the following sections.

```
{
  data: [{
      name:      "Max Mustermann",
      birthDate: "15.03.2016"
  }]
}
```

Listing 1: Example source data

```
{
  data: [{
      name: {
        value:       "Max Mustermann",
        key:         "name",
        attributes: []
      },
      birthDate: {
        value:       "15.03.2016",
        key:         "birthDate",
        attributes: []
      }
  }]
}
```

Listing 2: Restructured example source data

3.5 Processing JSON Data

The processing of JSON data follows two separate ways (cf. Figure 4). The first way is the processing of locally available data – either provided as textual input or an uploaded file. The second option describes the processing of resources to be accessed using an API endpoint. In that case a special API documentation processor will be invoked to gather metadata of the provided source data.

3.5.1 Extending JSON Tree

In order to store the matched attributes or possible predicates for every object attribute inside the JSON string (see Listing 1), we extend it with an empty placeholder array. Thereto, we iterate over every single literal and check if the

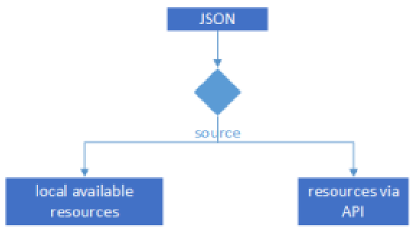


Figure 4: Differentiation of JSON input sources

type matches a string. In case of a match, the whole string gets wrapped inside a new object that contains the key-value pair and the mentioned empty attributes array (see Listing 2).

3.5.2 *Tracking Relevancy*

The algorithm relies on a large variety of rules or qualifiers that are used to identify the type or format of a string value. To measure if such a rule is generating proper results we propose a *Relevancy* value. This value is within the interval 0..1 and is adapted through time by evaluating if a rule has generated a correct mapping. Thus, the system can use the Relevancy value to learn and improve itself. This evaluation can be done in two different manners.

The first option is to manually check the results by the user. Therefore, the final result is displayed to the user and he is offered to adjust the displayed mappings according to his cognitional skills and knowledge. The adjusted result gets processed by the algorithm again and it will be checked whether the automatic decisions were correct or wrong. Based on this information, the Relevancy value of each stored and analyzed entry will be adapted.

The second option is to provide a pair of source data and the desired result. Thus, the algorithm can compare its own automatically generated result to the given one. This also enables to adapt the Relevancy value in order to improve later extractions.

```
{
    name: "Max Mustermann",
    birthDate: "15.03.2016"
}
```

Listing 3: Example source data as key-value pairs

```
{
    data: [
        "Max Mustermann",
        "15.03.2016"
    ]
}
```

Listing 4: Example source data as array structure

3.5.3 *Analyzing Data Structure*

This processing step of analyzing the data structure covers the validation of structure of the JSON resource by checking the existence of key-value-pairs within the objects (cf. Listing 3). This is important because the analysis of key names offers an additional way to correctly map predicates. It is also examined if values are encapsulated in an array and possess own attributes or if they are represented in a plain array (cf. Listing 4).

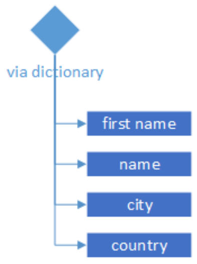


Figure 5: Analysis of values using dictionaries

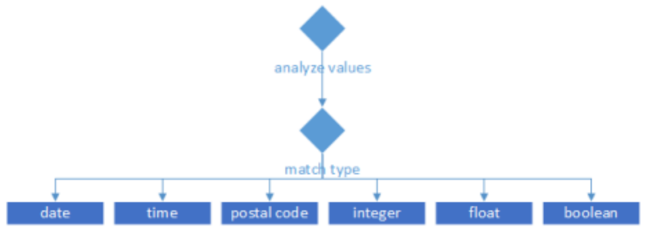


Figure 6: Analysis of the structure of values

3.5.4 *Analyzing Values*

Afterwards, the data values are analyzed in two subsequent steps. Firstly, the meaning of a value is identified by using existing dictionaries (cf. Figure 5). For instance, we try to look up a string value within a list of known first names. If this string is found, this may be a hint that the given string is actually a name. However, this value could concurrently exist in a list of last names, so that both options need to be considered. Therefore, we store all matches within the placeholder array of this value. The mostly likely match will be computed later together with all other found ones depending on their Relevancy value. To also identify values that may be combinations of known words, the proposed algorithm subsequently tries to match every part (separated by whitespace) on its own. An example would be a name that is a combination of a first and a last name.

The value strings are again analyzed in a second processing step. This time, we examine the string itself by using predefined templates to check for a certain format. An example could be a string consisting three numbers separated by a dot character, where two numbers are two digits and one is four digits long, which could be an indication for a date value. Those templates can be described using regular expressions. A good example would be that the identified date value is likely to be a “birthDate” within the context of a “http://schema.org/Person”. Thus, in summary this processing step is entirely based on predefined templates.

3.5.5 *Analyzing Keys*

The mapping of keys to predicates is also performed in a two-way fashion. The first step is the attempt to assign existing attributes to keys. In fact, it is possible that a key can be mapped to multiple attributes. Relevancy is again an important indicator. The second step is to search for synonyms of different keys in dictionaries, which are then mapped onto attributes. All found matches will be stored in the prepared attributes array as shown in Listing 5.

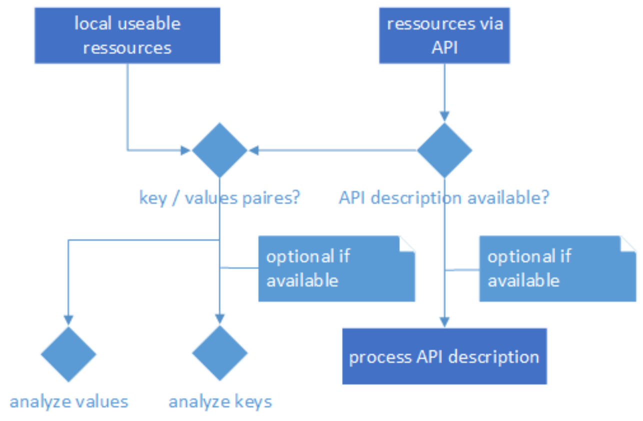


Figure 7: Analysis steps for local and remote resources

```
{
  data: [{
    name: {
      value:      "Max Mustermann",
      key:        "name",
      attributes: [...,{
          "id":        1,
          "value":     "name",
          "relevancy": 0.5,
          "type":      "analyzeValuesPerKey"
      }]
    },
    birthDate: {
      value:      "15.03.2016",
      key:        "birthDate",
      attributes: [...,{
          "id":        2,
          "value":     "birthDate",
          "relevancy": 0.5,
          "type":      "analyzeValuesPerKey"
      }]
    }
  }]
}
```

Listing 5: Annotated example data

3.5.6 *Processing API Descriptions*

Many web applications that provide an API also offer a service description. Thus, it should be possible to extract metadata of the source data by analyzing the API description for the specific entries (cf. Figure 7). Available example data can also be parsed and is helpful to obtain information about the structure of the data. The obtained information is also used to map appropriate predicates.

3.5.7 *Identifying Matching Predicates*

The task of identifying matching predicates unites the results of the preceding steps, which are the analysis of keys and values as well as the potential processing of an API description. From the set of suggested candidates that were obtained through the analysis the most-probable one will be chosen under consideration of its Relevancy value. With the selected one, the algorithm will then created a mapping of the property to the according predicate.

3.5.8 *Finding RDF Classes*

After the algorithm has selected matching predicates, a search for an appropriate RDF class for the data objects is performed. We aim to choose RDF classes where preferably all selected predicates belong to one RDF class. A suitable RDF class is then stored within the object with a generated id and a corresponding reference. Apart from that, RDF classes can also be obtained using special search engines or through manual user input.

```
[{
  "http://schema.org/person": [{
      "@id": "_:p1"
  }]
},{
  "@id": "_:p1",
  "http://schema.org/name": [{
    "@value" : "Max Mustermann"
  }],
  "http://schema.org/birthDate": [{
    "@value" : "15.03.2016"
  }]
}]
```

Listing 6: Resulting extracted JSON-LD

3.5.9 *Expanding JSON to RDF*

After enriching the JSON source data with the additional metadata as well as selecting appropriate RDF predicates, we proceed with the transformation of the data into a valid JSON-LD structure. As an example see Listing 6, where a person named “Max Mustermann” is described. For every person, an instance of the corresponding RDF class “http://schema.org/person” is referred and described with an unique identifier. Afterwards, all predicate-object pairs described by their URI and value are listed with an reference to the aforementioned instance identifier. Those steps are repeated for all persons found.

3.5.10 *Validating and Converting Results*

Finally, a validation in terms of correct syntax of the generated JSON-LD file is performed. The last step in the whole extraction process is the conversion into the configured target RDF format. Thereto, we use an existing conversion library (e.g., EasyRdf).

4. EVALUATION

To evaluate our approach, we have built a prototype that implements our algorithm as a webservice in node.js. The implementation contains all proposed components except the “Analyze API Description” component (cf. Figure 8). As a basic schema we use the schema.org RDF classes and according predicates. Therefore, we implemented a schema.org parser that imports the RDF classes and predicates from the schema.org website into our database. In Figure 9 a screenshot of our prototypical web application displaying the imported data is shown. Our evaluation scenario aims to extract of knowledge from a semi-structured list of people given as JSON using the schema.org/people class in a most sufficient way. This example data contains 1.099 attributes and objects describing 140 persons. The goal is to match as many as possible attributes to RDF predicates for every listed person.

In our evaluation environment we initialize the algorithm

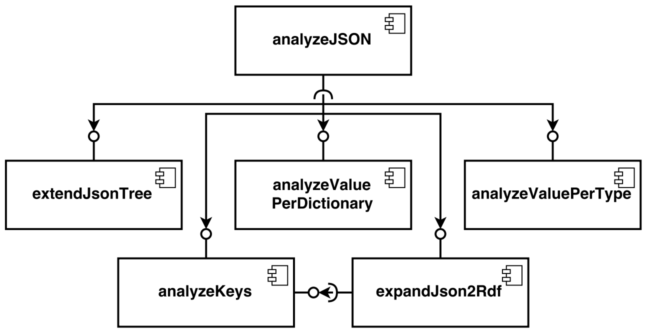


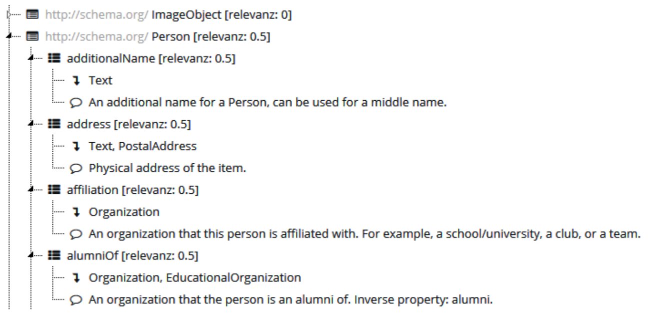
Figure 8: Components of the implementation

with the following configuration: The database contains a set of RDF classes with their associated predicates. We predefined the following four dictionaries that are used for the dictionary-based analysis component: first-names, last-names, streets and cities. All of them are initially set to a relevancy value of 0.5. The value-type analysis component uses four predefined templates. The first template is used to check whether a string can be parsed into a date value. The second template checks, if a string is a valid email address. They are by default mapped to the schema.org/person *birthDate* respectively *email*. The third template checks, if a string matches URL and the last one checks, if a string ends with a known file extension of an image. The key analysis as well as the predicate and class mapping has no initial configuration. With this initial setup, the KESeDa approach is able to distinguish all 140 people with in total a number of 518 mapped predicates. These result are highly depended on the configuration of the predefined templates. If the templates are well prepared, the algorithm will match a good set of predicates just from the beginning.

```
<rdf:Description rdf:nodeID="genid29">
 <schema:email
  rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  martin.seidel@informatik.tu-chemnitz.de</schema:email>
 <schema:firstName
  rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  Martin</schema:firstName>
 <schema:homepage
  rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  /images/people/MartinSeidel.png</schema:homepage>
 <schema:lastName
  rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  Seidel</schema:lastName>
</rdf:Description>
```

Listing 7: Example RDF result of a person

In the next evaluation step, the algorithm will learn associations between the schema.org vocabulary and the given properties from the JSON source. Thereto, we fed the algorithm with a single person containing all possible information and a resulting JSON object that has all properties correctly assigned to the according predicates. Using this information, the algorithm can check, if the automatically mapped predicates match the given ones. Following, the Relevancy value of every property is adjusted depending on if a correct mapping was selected. Furthermore, new mapping rules for predicates and associated attribute names are stored if no rule for that already existed. After this training, we again apply our algorithm on the example data. In comparison to the initial setup, we again were able to detect all 140 persons



```
http://schema.org/ ImageObject [relevanz: 0]
http://schema.org/ Person [relevanz: 0.5]
  additionalName [relevanz: 0.5]
    Text
    An additional name for a Person, can be used for a middle name.
  address [relevanz: 0.5]
    Text, PostalAddress
    Physical address of the item.
  affiliation [relevanz: 0.5]
    Organization
    An organization that this person is affiliated with. For example, a school/university, a club, or a team.
  alumniOf [relevanz: 0.5]
    Organization, EducationalOrganization
    An organization that the person is an alumni of. Inverse property: alumni.
```

Figure 9: Example import from schema.org

but now could map a total number of 679 predicates. All attributes that could be mapped to a schema.org/person predicate were correctly chosen. An excerpt of an exemplary result is displayed in Listing 7.

5. CONCLUSION

The paper gave an insight in the extraction process of semi-structured data from heterogeneous data sources. Underlying technologies, formats and definitions were discussed in detail. We conducted a detailed analysis of existing extraction solutions in Section 2. The review showed that some specific types of semi-structured data can already be processed and thus, existing tools can be utilized as part of a comprehensive extraction solution. But especially the data format JSON is not supported yet.

The paper focused on the development of a concept to process all kinds of semi-structured data formats (HTML, XML and JSON) for extracting knowledge from it. The approach was divided into several components in order to enhance its re-usability. In a first step, a format analysis was performed to find out the concrete data representation format. In the special case of JSON a new algorithm was designed on how to extract the therein contained knowledge. An examination of values through dictionaries and its data type as well as an analysis of its keys was used in order to infer certain predicates. From the set of potential predicates an RDF class for an object was derived and stored using the JSON-LD format.

This paper has shown that our new approach is capable of extracting knowledge directly from JSON-based data sources. The quality of resulting extraction depends on the configuration of the system. This includes the amount of templates for data type recognition and how suitable they are and also the distinctive linkage between these templates and predicates. The latter aspect can be improved by evolving the system learning processes.

In future work, we will conduct a more detailed evaluation of the solution. We want to analyze different learning methods and evaluate how the training from one data source will effect other sources.

6. ACKNOWLEDGMENT

This work was supported by the grant from the German Federal Ministry of Education and Research (BMBF) for the LEDS Project under grant agreement No 03WKCG11D.

References

[1] B. Adelberg. NoDoSE – A Tool for Semi-automatically Extracting Structured and Semistructured Data from Text Documents. *SIGMOD Rec.*, 27(2):283–294, June 1998. ISSN 0163-5808.

[2] M. Álvarez, A. Pan, J. Raposo, F. Cacheda, and A. Viña. FINDER: A Mediator System for Structured and Semi-structured Data Integration. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications, 2002.*, pages 847–851, Sept 2002.

[3] G. O. Arocena and A. O. Mendelzon. WebOQL: Restructuring Documents, Databases and Webs. In *Proceedings of the 14th International Conference on Data Engineering*, pages 24–33. IEEE, 1998.

[4] E. Baranovskiy. Methodik zur automatisierten Extraktion und Klassifikation semistrukturierter Produkt- und Adressdaten aus Webseiten. Master's thesis, Universität Stuttgart, 2011.

[5] R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web Information Extraction with Lixto. In *VLDB*, volume 1, pages 119–128, 2001.

[6] R. Baumgartner, G. Gottlob, and M. Herzog. Scalable Web Data Extraction for Online Market Intelligence. *Proceedings of the VLDB Endowment*, 2(2):1512–1523, Aug. 2009. ISSN 2150-8097.

[7] L. Bühmann, R. Usbeck, A.-C. N. Ngomo, M. Saleem, A. Both, V. Crescenzi, P. Merialdo, and D. Qiu. Web-Scale Extension of RDF Knowledge Bases from Templated Websites. In *The Semantic Web–ISWC 2014*, pages 66–81. Springer, 2014.

[8] L. Bühmann, R. Usbeck, A.-C. Ngonga Ngomo, M. Saleem, A. Both, V. Crescenzi, P. Merialdo, and D. Qiu. *Web-Scale Extension of RDF Knowledge Bases from Templated Websites*, pages 66–81. Springer International Publishing, 2014.

[9] F. Corcoglioniti, M. Rospocher, and A. P. Aprosio. A 2-phase Frame-based Knowledge Extraction Framework. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 354–361. ACM, 2016.

[10] E. Cortez and A. S. da Silva. Unsupervised Strategies for Information Extraction by Text Segmentation. In *Proceedings of the Fourth SIGMOD PhD Workshop on Innovative Database Research*, IDAR '10, pages 49–54, New York, NY, USA, 2010. ACM.

[11] V. Crescenzi and G. Mecca. Grammars have Exceptions. *Information Systems*, 23(8):539–565, 1998.

[12] V. Crescenzi, G. Mecca, P. Merialdo, et al. Roadrunner: Towards Automatic Data Extraction from Large Web Sites. In *VLDB*, volume 1, pages 109–118, 2001.

[13] C. Ebert. Das Kompositionalitätsprinzip, 2005.

[14] D. Freitag. Machine Learning for Information Extraction in Informal Domains. *Machine Learning*, 39:169–202, 2000.

[15] J. Hammer, J. McHugh, and H. Garcia-Molina. Semistructured Data: The TSIMMIS Experience. In *In Proceedings of the First East-European Workshop on Advances in Databases and Information Systems-ADBIS '97*, pages 1–8, 1997.

[16] C.-N. Hsu and M.-T. Dung. Generating Finite-state Transducers for Semi-structured Data Extraction from the Web. *Information Systems*, 23(8):521–538, 1998.

[17] G. Huck, P. Frankhauser, K. Aberer, and E. Neuhold. Jedi: Extracting and Synthesizing Information from the Web. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems*, pages 32–41. IEEE, 1998.

[18] S. Kara, Ö. Alan, O. Sabuncu, S. Akpınar, N. K. Cicekli, and F. N. Alpaslan. An Ontology-based Retrieval System Using Semantic Indexing. *Information Systems*, 37(4):294–305, June 2012. ISSN 0306-4379.

[19] N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997.

[20] N. Kushmerick. Wrapper Induction: Efficiency and Expressiveness. *Artificial Intelligence*, 118(1):15–68, 2000.

[21] A. H. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A Brief Survey of Web Data Extraction Tools. *ACM Sigmod Record*, 31(2):84–93, 2002.

[22] L. Liu, C. Pu, and W. Han. XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources. In *Proceedings of the 16th International Conference on Data Engineering*, pages 611–621. IEEE, 2000.

[23] B. Ludäscher, R. Himmeröder, G. Lausen, W. May, and C. Schlepphorst. Managing Semistructured Data with Florid: A Deductive Object-oriented Perspective. *Information Systems*, 23(8):589–613, 1998.

[24] C. V. Monllaó. *Ontology-Based Information Extraction*. PhD thesis, Polytechnic University of Catalunya, 2011.

[25] R. Mooney. Relational Learning of Pattern-match Rules for Information Extraction. In *Proceedings of the 16th National Conference on Artificial Intelligence*, 1999.

[26] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical Wrapper Induction for Semistructured Information Sources. *Autonomous Agents and Multi-Agent Systems*, 4(1-2):93–114, Mar. 2001. ISSN 1387-2532.

[27] W. Petersen. Einführung in die Computerlinguistik – Part-of-Speech-Tagging, 2004.

[28] A. Sahuguet and F. Azavant. Building Intelligent Web Applications using Lightweight Wrappers. *Data & Knowledge Engineering*, 36(3):283–316, 2001.

[29] S. Soderland. Learning Information Extraction Rules for Semi-structured and Free Text. *Machine Learning*, 34:233–272, 1999.

[30] M. Völkel. Extraktion von XML aus HTML-Seiten. Master's thesis, Diplomarbeit, Universität Karlsruhe, IPD Goos, 2003.

[31] Y. Wilks and M. Stevenson. Sense Tagging: Semantic Tagging with a Lexicon. *arXiv preprint cmp-lg/9705016*, 1997.