Semantic integration for food vending systems

Miguel PRAGOSA$^*$ and Vitor FERNANDES$^*$

$^*$Polytechnic Institute of Leiria, School of Technology and Management, Campus 2, 2411-901 Leiria, Portugal

Abstract. The adoption of new technologies by some areas that were not expected to use it may lead to the need of integrating heterogeneous systems. This paper intends to demonstrate that, in some cases, the use of the semantic techniques is the only way to integrate those systems. To do this we used several lexical and grammatical integration techniques and we focused on the integration of two food databases. The system developed may be used by a large number of areas. This is highly customizable and is able to provide the expected information with a good accuracy level.

Keywords. semantic web, search engine, ontologies, system integration

Introduction

During the last years, the information technologies reached a lot of new areas, which were not expected to use it. This may lead us to the need to integrate heterogeneous systems which are not ready to interact. Additionally, the implementation of data exchange rules in heterogeneous systems is a very rare scenario.

Given this, we can consider that the implementation of semantic techniques is sometimes the only way to ingrate systems which are not ready to communicate.

To show the concept above, this document will focus on the integration of a food vending system database with a central food repository. Now, and for test purposes, we will focus on the Portuguese language only. Later we intend to adapt our work to other languages. The main target of this work is to match each of the Point of Sale (POS) items with a central database (DB) item.

In practice, this process will provide a unified record of the food consumption data. This will allow us to implement several different applications, for example to create consumption profiles based on the nutritional values of the items sold.

In order to fit those targets, we used several lexical and grammatical tools as well as ontology techniques. Those techniques will be further addressed during this paper.

The software development was made using the Java programming language and the Hibernate framework. We also used the MySQL Database Management System (DBMS) as the item data store.

In this paper we first address the semantic integration background. The following section presents related work that was considered interesting. Section three describes the resource analysis and also the customizations made. Section four describes the attribute extraction process, and section five shows the comparison engine. Finally, section six shows the evaluation results and section seven concludes this paper.
1. Related work

This section aims to inform the reader about the efforts already made in order to achieve the targets defined by this document.

Given the scope of our work, this section will focus on two business areas: The semantic search engines, since that is where our solution basis, and the semantic food repository systems, since those kinds of tools can show us some the assets of applying the Web semantics to the food systems.

1.1. Search engines

The search engines are one of the applications which can become more intelligent with the aid of the semantic tools. After that evolution we can expect that the search engines will be able to understand, not only known keywords, but also complex human-readable sentences.

The Kngine\(^1\), the Hakia Search Engine\(^2\) or the DuckDuckGo\(^3\) are examples of search engines which already take advantage of the Web semantic technologies. This allows them to provide a broader and a most intelligent set of results to its users.

In July 2010, Google acquired a startup company focused on the development of an open and really comprehensive semantic database – the “Freebase”. Given this, and despite the lack of official information, we could expect the Google generic search engine to include some semantic features in the next few months.

1.2. Food repositories

Since January 2011 Google uses the Web semantics technologies in order to provide a recipe searching engine – The “Google Recipe View” [1]. This tool is currently available only in the United States of America (USA) and Japan.

This search engine allows its users to search by a certain ingredient which should or should not be contained on the results retrieved. This tool also allows the user to filter the list of items returned by the cooking time, by the number of calories, etc.

The Yummly\(^4\) appears as a good alternative, as it offers a lot of additional functionalities. Among other features, it is possible to filter the result list based on the desired taste (more or less sweeter, spicy, etc.), on a set of allergies and diets, etc.

Finally, it is also important to talk about the Wordnet project. Started on the University of Princeton (on the USA), that project aims to establish a relational network between the members of a language. This network is built based on relations like the equivalence or the similarity.

Nowadays most of the countries have its own Wordnet. For instance, the Portuguese Wordnet – “Wordnet.PT” – currently offers a total of 19,000 expressions divided into several semantic domains [2]. This Wordnet implementation was made by the Computation of Lexical and Grammatical Knowledge Research Group of the University of Lisbon.

\(^1\) http://www.kngine.com/
\(^2\) http://www.hakia.com/
\(^3\) http://www.duckduckgo.com/
\(^4\) http://www.yummly.com/
2. Resource analysis and customization

Given the focus of this work (semantic integration of multiple systems), it is very important to do a deep analysis on the datasets of both applications. This way we will be able to make our solution more suitable for the data involved.

Some of the guidelines below must be taken as preconditions for the search engine to reach the results describer on this document.

2.1. Dataset analysis

As we already described above, this work basis on two different databases which will feed the integration engine.

The first one is a Point of Sale (POS) database which contains a total of 1457 items. This database only provides the name of each item.

The second dataset will be used as the central food repository and was provided by the Portuguese National Institute of Health “Dr. Ricardo Jorge”. This database provides us not only the item name, but also a set of around 40 different nutritional values for each product (such as the energy, the proteins or the fat contained). The dataset comprises of 972 distinct products.

2.2. Manual dataset normalization

After a deeper analysis on the both datasets, it was possible to define list of 60 non-food items taken from the POS database. As those items do not suit to our need, they were removed.

After that, we identified a total amount of 23 items whose name was truncated, so we decided to complete them. Some of the items also include improperly formed special characters. Those characters were replaced by the right ones.

We also found that 29 of the 972 products from the POS database had spelling errors, so we corrected those names. Then we concluded that the nationality of the POS products was written in different ways, so it could be possible for a machine to understand its meaning. Given this, we decided to apply the standard ISO 3166-1 alpha-2 [3], which defines a two-letter code for each country, dependent territory and areas of geographical interest. This customization was applied over 35 products.

Finally, we also removed all the words that may deteriorate the trust level provided by the built engine (e.g., some of the products contained its own lexical category on the beginning of its name).

All the topics referred above must be taken into account when adding new items into the datasets. Otherwise, the accuracy level may not be the ideal one.

2.3. Automatic dataset normalization

Beyond the manual normalization process, we also decided to develop an automatic and more flexible way to do the dataset customizations. This process aims to replace some abbreviations or shot forms and also to remove some useless information contained on the datasets.

In practical terms, this engine basis on a text manipulation mechanism that is able to understand regular expressions. Each rule contains a regular expression, which
specifies the piece of text to search for and then to replace, and a text expression, which specifies the replacing text.

Due the adaptability requirements of our solution, the rules must be totally independent of the application, so we decided to store them on a set of Extensible Markup Language (XML) files. In order to allow the user to decide in which step will each rule be executed, the XML files will be handled in alphabetic order, and then each rule will be applied sequentially.

The application will consider all the XML files inside a predefined folder which could be validated against a custom XML Schema Definition (XSD) Model.

3. Attribute extraction

We can define the attribute extraction process as a way to enrich the words we want to act as the integration keys, or, in other words, a way to turn a meaningless word in a meaningful set of attributes, so that the application can understand them.

Those properties are the most valuable party during the compare process, since they will feed the search engine.

Along this section we will describe the methods we used in order to extract as much properties as we can.

3.1. Lexical analysis

The lexicon of a language represents its vocabulary, so during this work we will consider the lexicon as the way to define the meaning of each word (or set of words).

This process may address one or more words a time. In fact, this process will be applied over the smallest meaningful element we could identify inside an item name (or “term”). These elements may be called “tokens”.

The component of software responsible for the lexical analysis is often called “lexer”, “scanner” or “lexical analyzer”. During this work will call it “lexer” or “lexical classification engine”

This method only considers the meaning of each token individually, that is, ignoring the meaning that the token may have when placed next to the others.

Despite we need to classify the smaller elements of each term, this process does not necessarily mean to split a term based on a reference character (like a blank space or a hyphen). For example, if we consider the term “Earl gray tea”, the lexer must identify two tokens: “Earl gray”, a type of tea, and “tea”, a drink.

In order to know the words to identify and the meanings to assign, the lexer needs a dictionary that suits the dataset elements. To build this dictionary we considered all the 2181 terms resulting from the dataset normalization step. As the result we built a total of 4924 lexical rules.

3.2. Grammatical analysis

The grammar is part of every language, since it defines the way the words make sense together as a sentence. In other words, this defines the structure of a language.

The grammatical classes available may vary from language to language. For instance, the Portuguese language has 10 grammatical classes (nouns, pronouns, verbs, etc.), however the adoption of these classes does not suit our needs, since we are not
dealing with regular sentences, but with food product names. That way decided to consider the following grammatical classes:

- Product;
- Attribute;
- Recipe;
- Taste;
- Product type;
- Non-existing product;
- Finality.

That way we can identify not only a token’s meaning but also the function it represents on the product name.

Additionally we also built a classification mechanism which does a second-stage analysis. This process uses a set of keywords from a language (like the adverbs or the pronouns), and then it applies its rules over the words around them. In summary, this mechanism uses the language rules to classify some tokens. During this document we will call this engine “relational grammar engine”.

That engine also basis on a set of XML dictionaries. The XML element where the grammatical meanings are stored is called “grammaticalMeaning”.

Eggs (product)  with - potatoes (product) (without - peel) Non-existing product

Fig. 1. An example of a term together with the grammatical meaning of each token

Considering the example on figure Fig. 1, the relational grammar engine will identify the second and the fourth tokens as references. Then it will classify the token “potatoes” as a product (because it appears immediately after the word “with”), and the token “peel” as a non-existing product (because it appears immediately after the word “without”). If the tokens have already been classified, its classification will be dropped rather than the newest one.

Besides the grammatical classification, this process also adds a reference for the token to which each classified token refers (when applicable). This is mostly useful when dealing with tokens classified as an attribute, a recipe, a taste, a product type or a finality. This information will be stored on an XML element called “refersTo”.

The element “refersTo” as well as the “grammaticalMeaning” may use a special syntax in order to define its target (that syntax is not mandatory for the “grammaticalMeaning” field, since the user can simply specify the grammatical meaning that must be applied). This special syntax allows the rule to dynamically refer other tokens based on the distance from the current one, on its cardinality or even in its significance (or priority) inside the term.
Fig. 2. The special syntax which may be used to define the elements “grammaticalMeaning” and “refersTo”.

The figure above is a graphical representation of the dynamic syntax. That syntax can be represented according to the Extended Backus-Naur Form (eBNF) format:

```
```

Using this syntax we can specify expressions like “the second product identified”, “the last token identified before a given character index”, “the most important product identified”, among others.

3.3. Ontologies

In some cases the lexical and the grammatical classifications are not sufficient to define the terms as needed. Given this we decided to build a classification method based on ontologies.

An ontology is a data model which is able to represent a knowledge domain. The ontology will add some information to the tokens in order to contextualizing them against the others. That way, our application will be able to infer attributes and relations which are not present on the product names. Those elements could be very useful to compare the items (for example, to compare recipes).

Our application is ready to handle three different relations between the items: “subClassOf”, which defines the hierarchical relationship between two tokens, “sameAs”, which defines equivalence, and “contains”, which defines whether one token contains another.

The ontology file is automatically generated and does not need any user interaction. In order to keep all the rules centralized, the rules used by the ontology engine are stored on the dictionary XML files.

The ontology individuals are identified using regular expressions, which allow us to identify the items even if they vary in number or gender. This is a very important topic, since we cannot control how the items are written on both databases.

During the implementation of this engine we used the Web Ontology Language 2 (OWL2) [5], standard which basis on the Resource Description Framework (RDF) metadata data model [6]. These are open standards which basis on XML.

The integration between the developed application and the data files was made with the aid of the framework “The OWL API” (version 3.2.4 of July 2011) [7]. Unlike other alternatives like the “Jena RDF API” [8] (which only supports the RDF standard), that tool allow us to handle OWL2 files.
Additionally, we also used an application called Protégé (version 4.1.0 of June 2011) [9]. This open-source tool helped us analyzing the OWL documents, since it provides them through a graphical representation.

4. Comparison engine

Now that we already enriched the database items using the tools described above, we will present the comparison engine. This engine will handle all the attributes previously inferred in order to establish relations between them. After all, each comparison result will get a numeric score. The pseudocode below shows how the comparison engine processes the items from both databases.

```
For each POS DB product
  Get weights to apply
  For each Central DB product
    Quantify product relation
      If relation is greater than zero
        Store the result
      Else
        For each grammatical class
          For each POS DB token
            For each Central DB token
              Quantify token relation
                If relation is greater than zero
                  Store the result
  end.
end.
```

The steps in bold are the key steps and they will be described later on this section. During section 5.2 we will also describe the scoring process.

4.1. Processing algorithm

As we already described, this comparison engine will handle all the items from the POS database in order to find which of the central database items best suits to it. This process is divided into two stages: first, the application will compare the POS and the central DB items as they are shown (with no need to use the tokens identified by the classification process), then, if that comparison did not return any result, it will compare the tokens from both items.

The token comparison algorithm is described below. The product comparison works on a very similar way, since it is composed by the first three steps from the token comparison engine. In order to simplify the representation, please note that the POS token will be referred to as “token_A” and the central DB token will be referred to as “token_B”.

```
If token_A equals to token_B
  return 1
Else
  If normalized token_A equals to normalized token_B
    return 1 * NORMALIZATION_RATIO
  Else
    If root of token_A equals to root of token_B
```
return 1 * STEMMER_RATIO
Else
    If token A is same OWL individual than token_B
        return 1 * OWL_SAMEAS_RATIO
    Else
        Set score to 0
        If one OWL individual contains another
            score += 1 * OWL_CONTAINS_RATIO
        If tokens are sibling OWL individuals
            score += 1 * OWL_SIBLING_RATIO
        If token A equals to token_B’s OWL parent
            dist = getDistanceToBParent
            score += (1/dist) * OWL_EQ_PARENT_RATIO
        If there is a relation between the parents
            dist = getDistanceBetweenParents
            score += (1/dist) * OWL_PARENT_REL_RATIO
        return score
end.

In summary, the token comparison engine must quantify each comparison as a ratio between 0 and 1. The rest of the scoring algorithm will be described later.

The comparison process comprises of two main parts: the first one, where we will check if the tokens mean the same, and the second one, where we will find if the tokens share other kind of relations (like the hierarchical relationship).

The first step compares both tokens as they are, so in case of success it will return the best possible ratio – 1.

Then it will compare the tokens after submitting them to a normalization process. This process aims to replace all the special characters existing on the token. As this may cause a decrease on the accuracy of the algorithm, the trust level is multiplied by a ratio. This ratio is and must be lower than 1.

The third step works on a similar way, however it uses the root of each token to do the comparison. The trust level will also be multiplied by a ratio lower than 1.

The root of each word is provided by a stemmer – a lexical tool which manipulates a word in order to find the bigger invariable part of it. Our system is ready to use three different stemmers that suits to the Portuguese language: the Porter stemmer [10], the Orengo stemmer [11] and the Savoy stemmer [12][13].

The last step of the first part aims to check if the tokens are synonyms. This step uses the “sameAs” relations defined on the ontology previously created. Here we will also need to multiply the trust level by a ratio lower than 1.

The second part of the algorithm consists of four sequential steps. On the first one we will check if one token contains another, then we will check if the tokens are sibling individuals. On the third step we will check if one token is equals to the other’s parent, and finally we will check what is the hierarchical relation between the token’s parents.

All the second part steps, as well as the ontology engine could be enabled or disabled by the user. The ratios are also customizable by the user.

4.2. Scoring engine

The scoring engine is one of the most important parts of a search engine, since it defines the success rate of the algorithm. Give this, it is very import to build a reliable scoring engine.
In order to ensure that we can trust on the results provided, it is also very important to build an absolute scoring engine, so that the users can evaluate the real success rate of each execution. This means that each execution must be evaluated using an absolute scale (like a percentage) and that the comparison results must be scored proportionally.

To do this we calculate the best possible result of each comparison, and then we multiply this value by the ratio returned by the processing algorithm. After that we store both values. When all the comparisons of each token are completed, we will have the sum of all the best possible results as well as the sum of all the results obtained, so we can calculate a trust percentage.

5. Evaluation

We can easily use some quantitative metrics to measure the quality of the comparison engine. However we will also need a comparison basis to do that.

Given this, we had to establish a relation between the items of both databases, so that those links could be taken as the ideal results. It is also important to note that some POS products do not correspond to any of the central DB items, so the algorithm must ignore them. In summary we identified 615 items to be compared.

Once the processing engine finds a match for each of the POS product, the analysis engine will check the nutritional values of both of the expected and the found items. Then it will calculate the average deviation between the nutritional values of both items. Finally it will determine the global percentage of deviation (Table 1).

Table 1. An example of how the results evaluation is done: Item A has an average deviation of 28% and Item B has an average deviation of 0%. This results on a global deviation of 14%.

<table>
<thead>
<tr>
<th>Item</th>
<th>Nutritional value</th>
<th>Expected value</th>
<th>Found value</th>
<th>Deviation (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item A</td>
<td>Energy</td>
<td>150</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Fat</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Proteins</td>
<td>40</td>
<td>30</td>
<td>33</td>
</tr>
<tr>
<td>Item B</td>
<td>Energy</td>
<td>30</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Fat</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Proteins</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

Along with this method we will also use another metric which will let us know how many matches were successfully found by the algorithm.

5.1. Multi-objective optimization

This system, like any search engine, basis on several weights defined by the user or by the programmer. In order to make our system more flexible, we defined all of those values on an XML file, so that they could be changed by the user with no need to change the code.

Since those values are set manually, there is no way to ensure that they are the best ones. Given this, we built an optimization algorithm designed to maximize the number of matches and to minimize the deviation between the nutritional values of the expected products and the products found (those metrics will be further address during the next section).

To do this we used the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [14], since it has a large community supporting it, what allows us to have a larger knowledge
base available around its application. As the developed application basis on the Java programming language, we used the jMetal framework [15], which implements several multi-objective optimization algorithms, such as the SPEA2.

The optimization engine implemented basis on the XML files where the custom weights are defined. In order to make it more flexible, we decided to provide the main application as a jar file, so that the optimization engine could execute the search process as an Application Programming Interface (API).

In order to find the best weight values for the datasets addressed we instructed the optimization engine to do 100 executions of the search mechanism for each process.

5.2. Obtained results

After completing the development of the evaluation engine we were able to evaluate the obtained results. In summary our algorithm is able to find 57% of the static relations defined. Additionally, the expected core DB item is one of the three first search results for 83% of the POS items. It is also important to note that only 0.8% of the POS items are not related to the core DB item expected at all.

Since this work focus on the integration of food databases, it is very important to evaluate, not only the number of successfully matches, but also its accuracy in terms of the nutritional values.

In other words, it is more important for us to have a lower deviation between the expected nutritional values and the nutritional values found than to have a higher success rate in terms of matches.

As we already said, the central repository provides us around 40 different nutritional values for each product, so those are the attributes that we need to evaluate.

![Comparison between expected values and values found](image)

**Nutritional attribute name and unit**

- **Expected values**
- **Values found**
- **Average deviation (%)**

**Fig. 3.** Comparison between the expected values and the values found by the system
As we can see through the chart above, each pair of expected value and value found almost coincides. The green triangles show the average percentage of deviation between the two values.

Checking the average deviation we can demonstrate what was said above, since only one of the 43 attributes has an average deviation higher than 25%. Those values can be further analysed through the table 2.

Table 2. Comparison between the expected values and the values found by the system

<table>
<thead>
<tr>
<th>Nutritional attribute</th>
<th>Expected value</th>
<th>Value found</th>
<th>Average deviation (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy (kcal)</td>
<td>234,949</td>
<td>234,443</td>
<td>0,22%</td>
</tr>
<tr>
<td>Energy (kJ)</td>
<td>963,403</td>
<td>961,331</td>
<td>0,22%</td>
</tr>
<tr>
<td>Water (g)</td>
<td>54,522</td>
<td>53,099</td>
<td>2,61%</td>
</tr>
<tr>
<td>Protein (g)</td>
<td>8,140</td>
<td>8,053</td>
<td>1,07%</td>
</tr>
<tr>
<td>Fat, total (g)</td>
<td>13,710</td>
<td>13,374</td>
<td>2,45%</td>
</tr>
<tr>
<td>Carbohydrates (g)</td>
<td>31,062</td>
<td>32,815</td>
<td>5,64%</td>
</tr>
<tr>
<td>Carbohydrates (mono) (MSE)</td>
<td>33,012</td>
<td>34,866</td>
<td>5,62%</td>
</tr>
<tr>
<td>Mono + disaccharides (g)</td>
<td>20,069</td>
<td>21,347</td>
<td>6,37%</td>
</tr>
<tr>
<td>Organic acids (g)</td>
<td>0,696</td>
<td>0,697</td>
<td>0,09%</td>
</tr>
<tr>
<td>Alcohol (g)</td>
<td>11,396</td>
<td>11,193</td>
<td>1,77%</td>
</tr>
<tr>
<td>Starch, total (g)</td>
<td>23,461</td>
<td>25,345</td>
<td>8,03%</td>
</tr>
<tr>
<td>Oligosaccharides (g)</td>
<td>0,848</td>
<td>0,858</td>
<td>1,18%</td>
</tr>
<tr>
<td>Fibre, total dietary (g)</td>
<td>2,842</td>
<td>4,630</td>
<td>62,94%</td>
</tr>
<tr>
<td>Fatty acids, saturated (g)</td>
<td>6,611</td>
<td>6,225</td>
<td>5,83%</td>
</tr>
<tr>
<td>Fatty acids, mono. (g)</td>
<td>5,417</td>
<td>5,469</td>
<td>0,95%</td>
</tr>
<tr>
<td>Fatty acids, poly. (g)</td>
<td>1,813</td>
<td>2,083</td>
<td>14,90%</td>
</tr>
<tr>
<td>Fatty acids, trans (g)</td>
<td>0,764</td>
<td>0,706</td>
<td>7,60%</td>
</tr>
<tr>
<td>Linoleic acid (g)</td>
<td>1,827</td>
<td>2,106</td>
<td>15,28%</td>
</tr>
<tr>
<td>Cholesterol (mg)</td>
<td>77,861</td>
<td>75,206</td>
<td>3,41%</td>
</tr>
<tr>
<td>Retinol (pref. vit. A) (mg)</td>
<td>0,084</td>
<td>0,095</td>
<td>13,95%</td>
</tr>
<tr>
<td>Vitamin A (retinol equiv.) (RE)</td>
<td>83,621</td>
<td>95,284</td>
<td>13,95%</td>
</tr>
<tr>
<td>Carotene (mg)</td>
<td>327,734</td>
<td>336,842</td>
<td>2,78%</td>
</tr>
<tr>
<td>Vitamin D (ug)</td>
<td>1,317</td>
<td>1,418</td>
<td>7,67%</td>
</tr>
<tr>
<td>Alpha-tocopherol (mg)</td>
<td>1,203</td>
<td>1,315</td>
<td>9,29%</td>
</tr>
<tr>
<td>Thiamin (mg)</td>
<td>0,143</td>
<td>0,127</td>
<td>11,39%</td>
</tr>
<tr>
<td>Riboflavin (mg)</td>
<td>0,160</td>
<td>0,121</td>
<td>24,31%</td>
</tr>
<tr>
<td>Niacin equiv. (mg)</td>
<td>3,270</td>
<td>3,025</td>
<td>7,49%</td>
</tr>
<tr>
<td>Niacin, preformed (mg)</td>
<td>1,655</td>
<td>1,440</td>
<td>12,94%</td>
</tr>
<tr>
<td>Niacin equiv. (tryptophan) (mg)</td>
<td>1,916</td>
<td>1,862</td>
<td>2,79%</td>
</tr>
<tr>
<td>Vitamin B-6 (mg)</td>
<td>0,148</td>
<td>0,146</td>
<td>1,34%</td>
</tr>
<tr>
<td>Vitamin B-12 (ug)</td>
<td>0,957</td>
<td>1,021</td>
<td>6,68%</td>
</tr>
<tr>
<td>Vitamin C (mg)</td>
<td>22,917</td>
<td>22,965</td>
<td>0,21%</td>
</tr>
<tr>
<td>Folate (ug)</td>
<td>24,320</td>
<td>25,874</td>
<td>6,39%</td>
</tr>
<tr>
<td>Ash (g)</td>
<td>1,521</td>
<td>1,455</td>
<td>4,34%</td>
</tr>
<tr>
<td>Sodium (mg)</td>
<td>335,456</td>
<td>324,023</td>
<td>3,41%</td>
</tr>
<tr>
<td>Potassium (mg)</td>
<td>218,370</td>
<td>209,143</td>
<td>4,23%</td>
</tr>
<tr>
<td>Calcium (mg)</td>
<td>55,616</td>
<td>44,301</td>
<td>20,35%</td>
</tr>
<tr>
<td>Phosphorus (mg)</td>
<td>142,079</td>
<td>136,254</td>
<td>4,10%</td>
</tr>
<tr>
<td>Magnesium (mg)</td>
<td>24,258</td>
<td>27,295</td>
<td>12,52%</td>
</tr>
<tr>
<td>Iron (mg)</td>
<td>1,388</td>
<td>1,531</td>
<td>10,35%</td>
</tr>
<tr>
<td>Zinc (mg)</td>
<td>0,820</td>
<td>0,946</td>
<td>15,39%</td>
</tr>
<tr>
<td>Sucrose (g)</td>
<td>19,949</td>
<td>21,697</td>
<td>8,76%</td>
</tr>
<tr>
<td>Lactose (g)</td>
<td>3,508</td>
<td>3,689</td>
<td>5,18%</td>
</tr>
</tbody>
</table>

After calculating the global deviation value, we can conclude that our system is able to find the nutritional values expected with an average deviation of 8,28%.
6. Conclusions and Future Work

This paper presents a semantic search engine that is able to integrate heterogeneous systems which do not share any communication standard. As we already said, we believe this approach will widely adopted during the next years, since this could be the only way to integrate some isolated datasets.

As it could be verified through the last section, “Evaluation”, our system is able to reach the expected results with a low margin of error, which makes it reliable.

In fact, this is the most important target we defined for that project. Since that concept may be adopted by other scientific areas (like the medicine), this is very important to ensure the reliability of its results.

Another goal we already defined is the need for a high level of adaptability, for example, in terms of the datasets addressed. Our approach to address this topic was to implement all the user-customizable data (like the rules or the weights) using open standards, like the XML format. That way it will be possible to update the datasets with no need to change the application.

In terms of the future work, there are three main topics that we want to refer. Two of those are related to the ontologies, since it is one of the most important components of a semantic integration tool.

As we are dealing with a well-defined knowledge domain (the food systems), it may be convenient to integrate our application with another existing knowledge bases in this area. To do this it will be necessary to enable the application to merge the ontologies created with other ones.

It will also be an asset to implement more ontology operators and to allow more powerful queries over this information. For example, we could use the Semantic Web Rule Language (SWRL) [16] to perform those queries, since it allows us to perform inference over OWL files.

The last topic is related to the impossibility to control the items on the datasets, so this may force us to deal, for example, with misspelled item names. Given this, it will be very beneficial for the system to provide a spelling-check tool, able to find and correct those misspelled words.

7. References


