Issue Tracking System based on Ontology and Semantic Similarity Computation

Habes Alkhraisat
Department of computer science
Al-Balqa Applied University
AL salt, Jordan

Abstract—A computer program is never truly finished; changes are a constant feature of computer program development, there are always something need to be added, redone, or fixed. Therefore, issue-tracking systems are widely used on the system development to keep track of reported issues. This paper proposes a new architecture for automated issue tracking system based on ontology and semantic similarity measure. The proposed architecture integrates several natural language techniques including vector space model, domain ontology, term-weighting, cosine similarity measure, and synonyms for semantic expansion. The proposed system searches for similar issue templates, which are characteristic of certain fields, and identifies similar issues in an automated way, possible experts and responses are extracted finally. The experimental results demonstrated the accuracy of the new architecture, the experiment result indicates that the accuracy reaches to 94%.

Keywords—issue tracking; ontology; similarity computation; vector space model

I. INTRODUCTION

Issue tracking systems are implemented as a part of integrated project management system. Software projects rely on issue tracking systems to direct corrective maintenance activity and to guide the maintenance activities of software developers. Users report symptoms of the issue along with related information, that include short or detailed textual descriptions of the issue, product and component that are affected by the issue, and how to reproduce. Developers then verify and fix the reported issues. There are often many reports that are received and thus developers would need to prioritize which reports are more important than others.

Issue tracking systems looks like a natural language information retrieval system that can be queried with natural language and return knowledge. Therefore the use of semantic knowledge in developing issue tracking systems improve its ability to semantically infer the similar issues. With cumulative information about issues collected by the issue tracking system over a period and with integrated semantic techniques, it is possible to build semantic issue tracking system that semantically searches for similar issues and links the knowledge for each issue.

This paper, proposes issue-tracking system with integrated semantic techniques for transforming issues content information into meaningful knowledge. The proposed system searches the knowledge documented to inferred semantically similar issues and recommended developers and the most similar files related to the reported issue.

The motivation of this work, for inference of the knowledge field and for recommendation of experts and files related to the reported issue, knowledge document includes not only the previously reported issues but also object-oriented mapping ontology, programmer-readable annotation, and developer experience. The main contribution of the paper is that, it proposes an issue tracking system that predicts similar issue in aa semantic way, possible experts, and possible program files related to issue.

II. RELATED WORK

“Who Knows about That Bug? Automatic Bug Report Assignment with a Vocabulary-Based Developer Expertise Model” [1] uses source code vocabulary to find the most applicable expert for a given bug tracking item. This approach takes long time to make proper recommendations.


“Expert Recommmender Systems in Practice: Evaluating Semi-automatic Profile Generation” [3] uses a client program which examines the documents within a folder and subfolder which was selected by the user and sends these examined word statistics to the server and compares it with other statistics.

[1], [2], and [3] have the problem that they are not sufficiently integrated into the task workflow a bug tracking and project management system.

III. SYSTEM ARCHITECTURE

The semantic approach for issue tracking system, proposed in this paper, consists of two main process: frontend process for handling the users’ issues and backend system for processing issues. Fig. 1 illustrates the main components for issues tracking system proposed in this paper. Frontend system handles the submitted issue and deal with the issue real-time processing. The backend is the platform for frontend processing, and mainly processes and maintains the issue database.
Fig. 1. Overview of issue tracking System Architecture

Fig. 2 shows the global architecture and necessary components to build the proposed system and clarifies the interaction between the system components. The system starts by receiving issue from the user, and finishes by providing the appropriate response for the reported issue, passing through the different phases.

The first component handles the job of prepossessing of issues, source code, and system documentations, which includes tokenization, stemming and keyword extraction. In the second component, the system applies the similarity computation, the similarity computation includes the semantic extension, feature vector generation and similarity measures. In the third component, the response to the submitted issues are extracted from the issues database, for response extraction the following are applied: confidence computation, response selection, and automatic return.

The scenario for an issue tracking system include the following:

1) The user reports an issue to the issue tracking system.
2) Next, in the issue Processing Module, the issue is rephrased by expanding the issue and passing it to the Issue Extraction Module.
3) The Information Retrieval component is used to retrieve the relevant issues, response, files, and developers based on the important keywords that appear in the issue.

IV. PREPROCESSING MODEL

Preprocessing model starts by tokenizing issues dataset, internal and external system documentations. The tokenization splits up the entire issues, user manual, and programmer annotation into a bag of words. For improving the performance of extracting module and to have exactly matching stems, stemming algorithm has been applied to the bag of words generated after the tokenization process [4]. As a final step, the preprocessing model removes stop words from the bag of words generated after the stemming process. English stop word list which is available online¹ is used for the removal of stop words from the stem word.

To illustrate the preprocessing model, let’s study the following issue:

**Issue**: There is a problem while uploading the staff personal image.

**Tokens**: there, is, a, problem, while, uploading, the, staff, personal, images.

**Keywords**: problem, uploading, staff, personal, image

**Stem**: problem, upload, staff, person, image

Each keyword is enhanced with the synonyms terms extracted from the ontology. Therefore, our system adds takes the benefits of the shared ontologies and enriches the keyword senses with senses extracted from their synonyms. For semantic extension and keyword enrichment, the synonyms of keywords have been extracted from Macmillan Dictionary². As example, it is possible to enrich the keywords of issue by extraction all synonyms of word “problem” and we get: “problem, difficulty, trouble”, and by extraction all synonyms of word “image”, and we get: “photo, picture, portrait”. The outcome of preprocessing model is a bag of words. The bag of words is then used to represent the issue numerically as vector.

V. ONTOLOGY MAPPING

Ontologies plays an important role in applications based on the semantic technologies. It consists of concepts, relationships between concepts, restrictions and is described in the ontological languages like Web Ontology Language (OWL).

OWL represents the rich and complex knowledge about things, groups of things, and relations between things.

In some sense, object-oriented representation looks like the ontological representation. In this paper, we used the Semantic framework for mapping object-oriented model to semantic web languages [5]. The relations between object-oriented elements and ontologies are described in table 1.

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Object Oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Class</td>
</tr>
<tr>
<td>Instance object</td>
<td>Object</td>
</tr>
<tr>
<td>Property</td>
<td>Attribute</td>
</tr>
<tr>
<td>Predicate</td>
<td>Attribute name</td>
</tr>
<tr>
<td>Object</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>

The following example illustrates the mapping process between class written in PHP language and OWL.

```php
class employee
{
    private id;
    private name;
    private personal_image;
}
```

```owl
<owl:Class rdf:id="staff">
    <owl:DataObjectProperty rdf:id="id">
        <owl:Value rdf:resource="employee">
            <rdfs:range rdf:resource="integer" />
        <rdfs:range rdf:resource="employee">
            <rdfs:range rdf:resource="employee">
                <owl:DataTypeProperty rdf:id="name">
                    <rdfs:domain rdf:resource="employee" />
                </owl:DataTypeProperty>
            <owl:DataObjectProperty rdf:id="personal_image">
                <owl:DataTypeProperty rdf:id="id">
                    <owl:Value rdf:resource="employee">
                        <rdfs:range rdf:resource="string" />
                    <rdfs:range rdf:resource="string" />
                </owl:DataTypeProperty>
            </owl:DataObjectProperty>
        </owl:DataObjectProperty>
    </owl:DataObjectProperty>
</owl:Class>
```

TABLE I. OBJECT-ORIENTED PARADIGM AND ONTOLOGIES MAPPING

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Object Oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Class</td>
</tr>
<tr>
<td>Instance object</td>
<td>Object</td>
</tr>
<tr>
<td>Property</td>
<td>Attribute</td>
</tr>
<tr>
<td>Predicate</td>
<td>Attribute name</td>
</tr>
<tr>
<td>Object</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>

VI. SIMILARITY COMPUTATION

Computing the similarity between user’s issues with both the issues and program annotations databases plays an important role in the automated issue tracking system.

Issue similarity refers to the similarity between the keyword set of given issue annotated by ontology and the pattern keyword set of issues, and instance keywords have been replaced with class keywords in the keyword set of ontology.

There are many computational models for text similarity such as, support vector machines (SVMs), neural network (NN), machine learning, K-Nearest Neighbor (KNN), and so on. In this paper, vector space model (VSM) has been applied for implementing the propose issue tracking system. The VSM was developed for the SMART information retrieval system [6]. VSMs perform well on tasks that involve measuring the similarity of meaning between words, phrases, and documents [7].

The idea of the VSM is to represent large collection of documents as a vector in a vector space. Using VSM the set of issues and queries are represented as m-dimensional vectors of identifiers in a common vector space, and the vectors are organized into a matrix. The row vectors of the matrix correspond to words and the column vectors correspond to

The frequency of a term

\[ \text{frequency factor} = \frac{\text{frequency of term in issue}}{\text{number of occurrences of term in all issues}} \]

and that term

\[ \text{inverse document frequency} = \log \left( \frac{N}{\text{number of documents containing term}} \right) \]

where

\[ N \]

is the number of unique terms. The vector space will then have m rows and n columns. The element \( w_{ij} \) in document vector space represents a non-binary weight of the \( i^{th} \) term \( k_i \) in the \( j^{th} \) issue \( l_j \). Let the weight \( w_{ij} \) associated with a pair \((k_i, l_j)\) be positive and non-binary, then the issue \( l_i \) and query \( Q \) are represented as vectors:

\[
\mathbf{t}_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,m})
\]

\[
\mathbf{q} = (w_{q,1}, w_{q,2}, \ldots, w_{q,m})
\]

where \( m \) is the number of feature terms.

The relevance of an issue to a query is given by the similarity of their vectors. The weight for terms in queries and issues are used in the computing degree of similarity. The most popular way to measure the similarity of two vectors is to compute their cosine. The cosine of the angle between issue vector \( \mathbf{t}_i \) and query vector \( \mathbf{q} \) in vector space models can be measured as follows:

\[
\text{sim}(\mathbf{t}_i, \mathbf{q}) = \cos(\theta) = \frac{\mathbf{t}_i \cdot \mathbf{q}}{||\mathbf{t}_i|| \cdot ||\mathbf{q}||} = \frac{\sum_{i=1}^{m} w_{i,j} \cdot w_{q,i}}{\sum_{i=1}^{m} w_{i,j}^2 \cdot \sum_{i=1}^{m} w_{q,i}^2}
\]

(1)

\[
\text{sim}(\mathbf{t}_i, \mathbf{q}) \]

varies from 0 to +1, the vector model ranks the documents according to their degree of similarity to the query, the \( \text{sim}(l_i, Q) = 1 \) when \( l_i = Q \), and \( \text{sim}(l_i, Q) = 0 \) when \( l_i \) shares no terms with \( Q \).

VII. TERM-WEIGHTING SCHEME

The success of vector space model lies in term-weighting scheme. It assigns more weight to surprising events and less weight to expected events. The term weighting for the vector space model has entirely been based on single term statistics. There are three main factors: term frequency factor, collection frequency factor and length normalization factor. All three factor are multiplied together to make the resulting term weight.

In VSM a weight is assigned to each term in a document depends on the number of occurrences of the term in the issue. The frequency of a term \( k_i \) inside a document \( l_j \) referred to as the term frequency \( tf \) factor and is given by:

\[
\text{tf}(k_i) = \frac{\text{freq}_{k_i,j}}{\max_j \text{freq}_{k_i,j}}
\]

(2)

Furthermore, the inverse of the frequency of a term \( k_i \) among the documents in the collection referred to as the inverse document frequency \( idf \). \( idf \) measures of rareness of a term across all documents. Assume there are \( N \) documents in the collection, and that term \( k_i \) occurs in \( df_i \) of them. Then \( idf \) factor of term \( k_i \) is essentially

\[
\text{idf}(k_i) = \log \left( \frac{N}{df_i} \right)
\]

(3)

Thus, the \( idf \) of a rare term is high, whereas the \( idf \) of a frequent term is likely to be low. The \( tf \) and \( idf \) are combined, to produce a composite weight schema for each term in each document, the resulted weight schema is called a Term frequency–inverse document frequency \( tf – idf \) scheme.
The main contribution of this paper is that it proposes an ontological semantic based issue tracking system. To achieve the system goals, the proposed system combines the Vector Space Models with domain ontology representing the issue and code vocabulary. To improve the semantic similarity and accuracy of system, each word is enhanced using the synonyms terms extracted from the ontology pool and keywords synonyms database. Therefore, the system takes advantage of the shared ontologies available on the Web and semantically enriches the keyword senses with senses extracted from their synonyms.

The experimental results indicate that the system reaches an accuracy of 94% based on test set of 240 issues and 5 experts. In future extensions, the accuracy of the proposed system would be compared with Jira bug tracking [10].

<table>
<thead>
<tr>
<th>TABLE II. EXPERIMENT RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Issue</strong></td>
</tr>
<tr>
<td>Recall</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
</tbody>
</table>

References


