Constructing the Knowledge Base for Cognitive IT Service Management

Qing Wang¹, Wubai Zhou¹, Chunqiu Zeng¹, Tao Li¹, Larisa Shwartz² and Genady Ya. Grabanrnik³

¹School of Computing and Information Science, Florida International University Miami, FL, USA
²Cognitive Service Management, IBM T.J. Watson Research Center Yorktown Heights, NY, USA
³Dept. Math & Computer Science, St. John’s University, Queens, NY, USA
Outline

➔ Introduction
➔ System Overview
➔ Approaches to construct the knowledge base
  ◆ Phrase Extraction Stage
  ◆ Knowledge Construction Stage
  ◆ Ticket Resolution Stage
➔ Experiment
➔ Conclusion and Future
Introduction: Background

➔ IT service providers are facing an increasingly intense competitive landscape and growing industry requirements.
➔ Software monitoring systems are designed to actively collect and signal event occurrence and, when necessary, automatically generate incident tickets.
➔ Solving these IT tickets is frequently a very labor-intensive process.
➔ Full automation of these service management processes are needed to target an ultimate goal of maintaining the highest possible quality of IT services.

Figure 1: Service Management System.
Introduction: Background

- **Monitor**: emits an event if the alert persists beyond a predefined duration.
- **Enterprise console**: determines whether to create an incident ticket.
- **IPC System**: collects the tickets and stores them in the ticket database.
- **Administrators**: perform problem determination, diagnosis, and resolution.
- **Enrichment Engine**: uses various data mining techniques to create, maintain, and apply knowledge base to maximize the automation of IT service management.

Figure 2: The overview of IT service management workflow.
Introduction: Motivation

Structured fields: inaccurate or incomplete generated by alarm systems

Unstructured text: written by system administrators in natural language. Potential knowledge includes:
1. What happened? Problem
2. What troubleshooting was done? Activity
3. What was the resolution? Action

Figure 3: A ticket in IT service management and its corresponding resolution are given.
Introduction: Challenge

➔ Challenge 1: Even in cases where the structured fields of a ticket are properly set, they either have small coverage or do not distinguish tickets well, and hence they contribute little information to the problem resolution.

➔ Challenge 2: The ambiguity brought by the free-form text in both ticket summary and resolution poses difficulty in problem inference, although more descriptive information is provided.

➔ Challenge 3: IT service management and particularly problem determination, diagnosis, and resolution require a large investment of manual effort by system administrators.

Figure 4: Ticket distribution with structure fields.
Our proposed integrated framework consists of three stages:

1. **Phrase Extraction Stage**
   - (a) Phrase Composing and Initial Summary Analysis Component
   - (b) Phrase Refining Component

2. **Knowledge Construction Stage**

3. **Ticket Resolution Stage**

**Figure 5:** An overview of the integrated framework.
1. Phrase Extraction Stage

➔ In this stage, our framework finds important domain-specific words and phrases (‘kernel’).
  ◆ Construct a domain-specific dictionary
    ● Mining the “hot” (repeated) pattern from unstructured text field.
    ● Refining these repeated phrases by diverse criteria filters (e.g., length, frequency, etc.).

➔ We process ticket data from account1 of three different accounts managed by IBM Global Services.

Figure 6: Numbers of Tickets and Distinct Resolutions.
1.1 Phrase Composing and Initial Summary Analysis

➔ Use StanfordNLPAnnotator for preprocess the tickets data.
➔ Build a domain dictionary by using Word-Level LZW compression algorithm.
➔ Calculate the frequency of the repeated phrases in tickets data by using Aho-Corasick algorithm.


Figure 7: Repeated pattern extraction and frequency estimation.
1.1 Phrase Composing and Initial Summary Analysis

➔ Word-Level Lempel-Ziv-Welch (WLZW)
  ◆ Seeks the trade-off between completeness and efficiency and attempts to find the longest n-gram with a repeated prefix
  ◆ Time complexity: O(n)

➔ Aho-Corasick algorithm
  ◆ Locate all occurrences of any of a finite number of keywords in a string of text.
  ◆ Consists of constructing a finite state pattern matching machine from the keywords and then using the pattern matching machine to process the text string in a single pass.
  ◆ Time complexity: O(n).
1.1 Phrase Composing and Initial Summary Analysis

➔ Assume we have a dictionary $D$ composing 
   “job failed due to plc issue,”
   “job failed due to database deadlock,”
   “job failed due to sql error,”
   “database connectivity,”
   “sql server,”
   “sql server memory”
   }.

➔ AC algorithm first constructs finite State Automaton for dictionary using a Trie.
➔ And then estimates the frequency of the phrases in the dictionary for a single pass.

1.2 Phrases Refining

In this stage, we apply two filters to the extracted repeated phrases allowing the omission of non-informative phrases.

➔ **Phrase Length & Frequency Filters (length > 20 & frequency >= 10)**
➔ **Part-Of-Speech Filter**

<table>
<thead>
<tr>
<th>Applied Filter</th>
<th>Left Phrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency Filter &gt;= 10</td>
<td>1117 items</td>
</tr>
<tr>
<td>Length Filter &gt; 20</td>
<td>613 items</td>
</tr>
<tr>
<td>PoSTag Filter</td>
<td>323 items</td>
</tr>
</tbody>
</table>

---

**Table I: Definition of technical term’s schemes.**

<table>
<thead>
<tr>
<th>Justeson-Katz Patterns</th>
<th>Penn Treebank Entity Patterns</th>
<th>Examples in Tickets</th>
</tr>
</thead>
<tbody>
<tr>
<td>A N</td>
<td>JJ NN[PS</td>
<td>PS]</td>
</tr>
<tr>
<td>N N</td>
<td>NN[PS</td>
<td>PS]</td>
</tr>
<tr>
<td>A A N</td>
<td>JJ JJ NN[PS</td>
<td>PS]</td>
</tr>
<tr>
<td>A N N</td>
<td>JJ NN[PS</td>
<td>PS] NN[PS</td>
</tr>
<tr>
<td>A N N</td>
<td>NN[PS</td>
<td>PS] JJ NN[PS</td>
</tr>
<tr>
<td>N N N</td>
<td>NN[PS</td>
<td>PS] NN[PS</td>
</tr>
<tr>
<td>N P N</td>
<td>NN[PS</td>
<td>PS] IN NN[PS</td>
</tr>
</tbody>
</table>

A: Adjective, N: Noun, P: Preposition

**Table II: Definition of action term’s schemes.**

<table>
<thead>
<tr>
<th>Penn Treebank Action Patterns</th>
<th>Examples in Tickets</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBD(G</td>
<td>N)</td>
</tr>
<tr>
<td>VBD: base form Verb, VBD: past tense Verb</td>
<td>affecting/circumventing, given/taken</td>
</tr>
<tr>
<td>VBG: gerund Verb, VBN: past participle Verb</td>
<td></td>
</tr>
</tbody>
</table>
2. Knowledge Construction Stage

In this stage, we first develop an ontology model, and then tag all the phrases of the generated dictionary with the defined classes.

➔ Build the ontology model
  ◆ Define classes
  ◆ Define relations

➔ Knowledge Archive
  ◆ Manually tag the important phrases in the dictionary with their most relevant defined classes.

![Ontology model depicting interactions among classes.](image)

<table>
<thead>
<tr>
<th>Class</th>
<th>Definition</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>Object that can be created/destroyed/replace</td>
<td>memory fault; database deadlock</td>
</tr>
<tr>
<td>Action</td>
<td>Requires creating/destroying an entity</td>
<td>restart; rerun; renew</td>
</tr>
<tr>
<td>Activity</td>
<td>Requires interacting with an entity</td>
<td>check; update; clean</td>
</tr>
<tr>
<td>Incident</td>
<td>State known to not have a problem</td>
<td>false alert; false positive</td>
</tr>
<tr>
<td>ProblemCondition</td>
<td>Describe the condition that causes a problem</td>
<td>offline; abended; failed</td>
</tr>
<tr>
<td>SupportTeam</td>
<td>Team that works on the problem</td>
<td>application team; databases team</td>
</tr>
</tbody>
</table>
2. Knowledge Construction Stage

➔ Initial Domain Knowledge Base:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Activity</th>
<th>Action</th>
<th>Problem Condition</th>
<th>Support Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>automated process</td>
<td>accept</td>
<td>reboot</td>
<td>abended</td>
<td>active directory team</td>
</tr>
<tr>
<td>actual start</td>
<td>accepted</td>
<td>renew</td>
<td>bad data</td>
<td>app team</td>
</tr>
<tr>
<td>additional connection</td>
<td>achieved</td>
<td>rerun</td>
<td>deactivated</td>
<td>application team</td>
</tr>
<tr>
<td>address information</td>
<td>acting</td>
<td>reran</td>
<td>disabled</td>
<td>aqepds team</td>
</tr>
<tr>
<td>afr end</td>
<td>add</td>
<td>reset</td>
<td>dropped</td>
<td>bazaarsvoice team</td>
</tr>
<tr>
<td>alert</td>
<td>added</td>
<td>restoring</td>
<td>expired</td>
<td>bmc team</td>
</tr>
<tr>
<td>alert imr</td>
<td>affecting</td>
<td>retransmit</td>
<td>fails</td>
<td>bsd team</td>
</tr>
<tr>
<td>alerts</td>
<td>affects</td>
<td>fixed</td>
<td>failed</td>
<td>bureau team</td>
</tr>
<tr>
<td>alphanumeric values</td>
<td>altered</td>
<td>restart</td>
<td>false alert</td>
<td>business team</td>
</tr>
<tr>
<td>amex</td>
<td>aligned</td>
<td>restarted</td>
<td>false positive</td>
<td>bwinfra team</td>
</tr>
<tr>
<td>api calls</td>
<td>allocate</td>
<td>renewed</td>
<td>human error</td>
<td>cdm team</td>
</tr>
<tr>
<td>application</td>
<td>allocated</td>
<td>fixed</td>
<td>not working</td>
<td>CDM/GLEUDBD team</td>
</tr>
<tr>
<td>application code</td>
<td>applied</td>
<td>fixing</td>
<td>offline</td>
<td>cmit team</td>
</tr>
<tr>
<td>application impact</td>
<td>assign</td>
<td>recycle</td>
<td>stopped</td>
<td>control m team</td>
</tr>
<tr>
<td>atm messages</td>
<td>assigned</td>
<td>recycled</td>
<td>unavailable</td>
<td>convergys team</td>
</tr>
<tr>
<td>audit</td>
<td>blocks</td>
<td>recycling</td>
<td>under threshold</td>
<td>csp team</td>
</tr>
<tr>
<td>audit log</td>
<td>bring</td>
<td>reopen</td>
<td>wrong</td>
<td>cu team</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Number of Tagged Phrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>628 items</td>
</tr>
<tr>
<td>Activity</td>
<td>243 items</td>
</tr>
<tr>
<td>Action</td>
<td>24 items</td>
</tr>
<tr>
<td>Problem Condition</td>
<td>22 items</td>
</tr>
<tr>
<td>SupportTeam</td>
<td>76 items</td>
</tr>
</tbody>
</table>
3. Ticket Resolution Stage

The goal of this stage is to recommend operational phrases for an incoming ticket.

➔ Information Inference component: This component is used to infer problems, activities and actions form the trouble ticket based on the constructed knowledge base and ontology model.

◆ Class Tagger Module: an index tool tagging the tickets in three steps.

- Defined Concept Patterns for Inference: concept patterns based on Problem, Activity and Action concepts.
- Problem, Activity and Action Extraction: Given the input tickets, the Class Tagger module outputs a list of tagged phrases and use concept patterns to extract the Problem, Activity and Action from the important tickets.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Pattern</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>Entity preceded/succeeded by ProblemCondition</td>
<td>(jvm) is (down)</td>
</tr>
<tr>
<td>Activity</td>
<td>Entity preceded/succeeded by Activity</td>
<td>(check) the (gift record count)</td>
</tr>
<tr>
<td>Action</td>
<td>Entity preceded/succeeded by Action</td>
<td>(restart) the (database)</td>
</tr>
</tbody>
</table>
3. Ticket Resolution Stage

The goal of this stage is to recommend operational phrases for an incoming ticket.

➔ **Ontology-based Resolution Recommendation component**
  
  ◆ Previous study, the KNN-based algorithm will be used to recommend the historical tickets’ resolution to the incoming ticket which have the top summary similarity scores.
  
  ◆ Jaccard similarity suffers bad performance due to many non-informative words.
  
  ◆ **Ontology model** can greatly facilitates our resolution recommendation task by **better capturing the similarity** between ticket summaries.
 Experiment

➔ Dataset
◆ Experimental tickets are collected from real production servers of IBM Cloud Monitoring system covers three month time period containing |D| = 22,423 tickets.
◆ Training data: 90% of total tickets
◆ Testing data: 10% of total tickets

➔ Evaluation Metrics
◆ Precision, Recall, F1 score and Accuracy.
◆ Accuracy = (TP + TN)/(TP + TN + FP + FN)
◆ Precision = TP/(TP + FP)   Recall = TP/(TP + FN)
◆ F1 score = 2 Precision Recall / (Precision + Recall)

TP = True Positive          TN = True Negative
FP = False Positive         FN = False Negative
Experiment

➔ **Ground Truth**
  - Domain experts manually find and tag all phrases instances into six predefined classes in testing dataset.

➔ **Evaluate our integrated system**
  - Class Tagger is applied to testing tickets to produce tagged phrases with predefined classes. Comparing the tagged phrases with ground truth, we obtain the performance.

![Figure 10: Evaluation of our integrated system.](image-url)
Experiment

➔ Evaluate Information Inference

◆ **Usability**: we evaluate the average accuracy to be 95.5%, 92.3%, and 86.2% for Problem, Activity, and Action respectively.

◆ **Readability**: we measure the time cost. Domain expert can be quicker to identify the Problem, Activity and Action which output from the Information Inference component from 50 randomly selected tickets.
Conclusion

⇒ Contribution
- A novel domain-specific approach.
- Utilization of the ontology modeling techniques.
- Automation improvement of IT service management.
- A closed feedback loop system for continuously extending of the knowledge base.

⇒ Future Work
- Investigate intelligent techniques to reduce human efforts on phrase tagging, such as training a conditional random field model.
- Leverage the ontology into Deep Learning model.
- Incorporate the obtained knowledge base into other tasks in the IT service management system.