Constructing the Knowledge Base for Cognitive IT Service Management

Qing Wang[1], Wubai Zhou[1], Chunqiu Zeng[1], Tao Li[1], Larisa Shwartz[2] and Genady Ya. Grabarnik[3]

[1]School of Computing and Information Sciences, Florida International University, Miami, FL, USA
[2]IBM T.J. Watson Research Center, Yorktown Hights, NY, USA
[3]Dept. Math & Computer Science, St. John's University, Queens, NY, USA

***Abstract*—The increasing complexity of IT environments dictates the usage of intelligent automation driven by cognitive technologies, aiming at providing higher quality and more complex services.**

Inspired by cognitive computing, an integrated framework is proposed for a problem resolution. In order to improve the efficiency of the problem resolution process, it is crucial to formalize problem records and discover relationships between elements of the records, records overall and other technical information. In the proposed framework, the domain knowledge is modeled using ontology. The key contribution of the framework is a novel domain specific approach for extracting useful phrases, that enables an automation improvement through resolution recommendation utilizing the ontology modeling technique. The effectiveness and efficiency of our framework are evaluated by an extensive empirical study of a large scale real ticket data.

***Keywords*-knowledge base; cognitive computing; ontology; IT service management;**

I. Introduction

A. Background

Driven by the rapid changes in the economic environment, business enterprises constantly evaluate their competitive position in the market and attempt to come up with innovative activities to gain competitive advantage. Value-creating activities cannot be accomplished without solid and continuous delivery of IT services. The increasing complexity of IT environments dictates the usage of cognitive incident management [15], one of the most critical processes in IT service management [1], [21], resolves the incident and restores the provision of services, while relying on monitoring or human intervention.

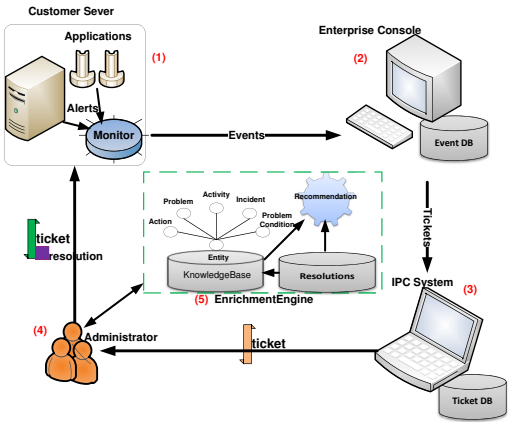


Figure 1: The overview of IT service management workflow.

A typical workflow of IT service management is illustrated in Fig. 1. It usually involves five steps. (1) As problems detected by a monitoring agent on a server, alerts are generated, and the monitoring emits an event if the alert persists beyond a predefined duration. (2) Events coming from an IT environment are consolidated in an enterprise console, which analyzes the monitoring events and determines whether to create an incident ticket for IT problem reporting. (3) Tickets are collected by IPC (Incident, Problem, and Change) system and stored in the ticket database [1]. (4) The system administrators perform the problem determination, diagnosis, and resolution based on the ticket description. The ticket resolution part of IT service delivery workflow is often a labor-intensive process. (5) In order to alleviate human efforts and maximize the automation of IT service management, the workflow incorporates an enrichment engine which in turn uses various data mining techniques to create, maintain and apply knowledge about the underlying IT system and its possible issues. The paper focuses on the construction of the knowledge base by processing ticketing information; it outlines an integrated solution that uses obtained knowledge to optimize problem resolution.

B. Motivation

STRUCTURED

TICKET IDENTIFIER:	WPPWA544:APPS:LogAdapter:NALAC:STARACTUAT_6600				
NODE	FAILURECODE	ORIGINAL SEVERITY	OSTYPE	COMPONET	CUSTOMER
WPPWA544	UNKNOWN	4	WIN2K3	APPLICATION	XXXX

UNSTRUCTURED

TICKET SUMMARY:	STARACTUAT_6600 03/01/2014 04:30:28 STARACTUAT_6600 GLACTUA Market=CAAirMiles:Report_ID=MRF600:ReportPeriod From: 2014/02/01 to 2014/02/28:ErrorDesc=For CAAirMiles Actuate is out of balance with STAR BalanceMRF600 & MRF601 Counts. Reconcilation Difference = 2MRF600 & MRF601 Net Fee. Reconcilation Difference = 25MRF600 & MRF601 Gross Fee .Reconcilation Difference = 25

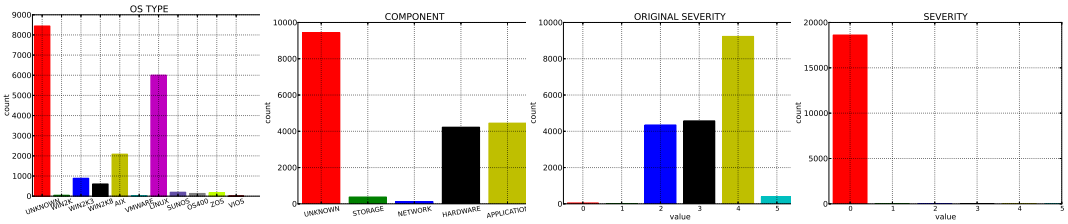
UNSTRUCTURED

RESOLUTION

ProblemSolutionText:***** Updated by GLACTUA ******
Problem Reported : Reconciliation difference Root cause : Reconciliation was run before all reports completed. This is as per the new SLAs.
Solution provided : *Reconciliation was re-run after the next set of reports completed.*There was no user impact.
Closure code : WRKS_AS_DSIGND
RCADescription:***** Updated by GLACTUA ******
Problem Reported : Reconciliation difference
Root cause : Reconciliation was run before all reports completed. This is as per the new SLAs.
Solution provided : Reconciliation was re-run after the next set of reports completed.There was no user impact.
Closure code : WRKS_AS_DSIGND

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

An example of an IT service management ticket is shown in Fig. 3. It consists of both structured fields (e.g., *OSTYPE*, *COMPONENT*) and unstructured free-form text fields (i.e., *SUMMARY* and *RESOLUTION*). Note that tickets are either generated automatically or reported by the system's user. The structured fields and the summary of a ticket provide the initial problem description for the system administrators (SAs) to start ticket resolution. SAs usually record the trou-



(a) OS Types. (b) Components. (c) Original severity. (d) Severity.

Figure 2: Ticket distribution with structured fields.

bleshooting steps in the resolution field as an unstructured free-form text.

In order to improve the efficiency of the problem resolution process, it is crucial to formalize the content of the ticket and, if possible, to discover a mapping between symptoms or a ticket's summary and resolutions. This is the initial motivation of our study. After a meticulous study and detailed analysis of the problem, a number of obstacles are identified.

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.

A subset of tickets are extracted from the historical ticket data set collected by IBM Global Services. Several fields such as *OSTYPE*, *COMPONENT*, and *SEVERITY* are investigated.

The distributions of the field values are shown in Fig. 2. As illustrated, the distributions are highly imbalanced in general. Specifically, most values of *OSTYPE* and *COMPONENT* fields are missing and labeled as *UNKNOWN*. We also observe that the field values such as *STORAGE*, *NETWORK*, *HARDWARE*, and *APPLICATION* only provide general information for problem type inference. Additionally, we provide the distributions of both original severity values generated by the monitoring and the severity values revised by human, denoted as *ORIGINAL SEVERITY* and *SEVERITY*, respectively. The severity values are considerably subjective since the two distributions of *SEVERITY* and *ORIGINAL SEVERITY* are extremely inconsistent.

Consequently, these structured fields are useful but by far not sufficient for precise problem inference. Thus we need to focus more on the free-form text fields in order to gain further insights into the underlying problem.

The analysis of free-form text fields reveals the following.

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.

Both ticket summary and resolution, illustrated in Fig. 3, contain domain-specific terms such as *SLAs*, *RCA*, and *WRKS_AS_DSIGND*. In addition they contain a number of typos and grammatical errors, such as *ErrorDesc* and *Problem-SolutionText*. Moreover, some text snippets may be repeated multiple times in a single ticket and resolution. An example is shown in Fig. 3 where phrases such as *Reconciliation Difference* and several other sentences appear in both ticket summary and resolution.

As a result, it becomes infeasible to identify useful information for problem inference using only traditional Natural Language Processing (NLP) techniques without any domain expertise.

As illustrated further, our proposed integrated framework is capable of gathering domain knowledge from logs, ticketing systems, and system administrators.

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

It is still a formidable task to fully automate the entire IT service management without the help of domain experts. Therefore, modeling, gathering, and utilizing the domain knowledge during ticket resolution become increasingly crucial.

In the proposed framework, the domain knowledge is modeled using ontology (see [3] for another application of ontology to IT Management) and organized into a knowledge base. In order to improve IT service management by making a number of steps toward its automation, a recommendation component leveraging the domain knowledge is explored to facilitate the ticket resolution.

C. Contribution

The contribution of our work mainly focuses on proposing and implementing an integrated framework that significantly improves the automation of IT service management. The key features of the proposed cognitive framework include:

- A novel domain-specific approach, designed to analyze free-form text in both ticket summary and resolution for useful phrase extraction.
- Utilization of the ontology modeling techniques, constructing a knowledge base by combining domain expertise with extracted useful phrases.
- Automation improvement of IT service management, through development of a resolution recommendation component based on domain knowledge.
- A closed feedback loop system, to facilitate learning from an outcome of resolution recommendation, and thus continuous extension of the knowledge base.

The effectiveness and efficiency of our framework are verified on a large data set of tickets from IBM Global Services.

The remainder of this paper is organized as follows. The overall framework is briefly introduced in Section II. The detail design and implementation of the proposed framework is provided in Section III. Section IV describes an extensive empirical study conducted over the real ticket data. The related work is presented in Section V. Section VI summarizes and concludes the paper.

II. System Overview

Taking the aforementioned challenges into account, an integrated framework is proposed. The framework is capable of constructing a knowledge base from discovered useful phrases mined from the tickets. It also incorporates

the domain knowledge provided by domain experts. The framework shows how the constructed knowledge base is used to optimize the IT service maintenance. The overall architecture of the integrated framework is illustrated in Fig. 4. Our proposed integrated framework consists of three stages: (1) Phrase Extraction, (2) Knowledge Construction, and (3) Ticket Resolution.

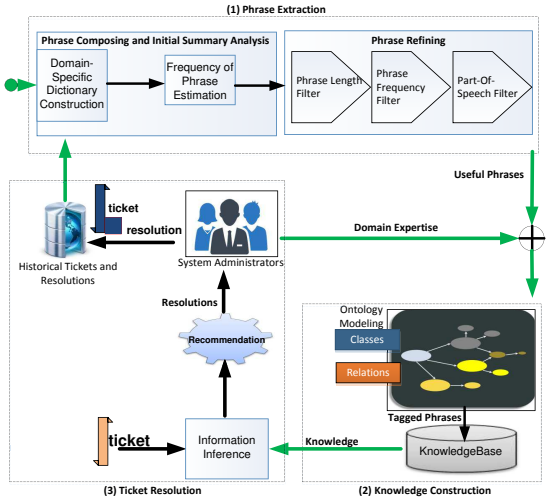


Figure 4: An overview of the integrated framework.

The entire framework starts with the stage of Phrase Extraction. The input of Phrase Extraction is a set of the historical tickets, and the output are the useful domain knowledge phrases. The Phrase Extraction stage involves two main components: the Phrase Composition and Initial Summary Analysis component, and the Phrase Refining component. The Phrase Composition and Initial Summary Analysis component builds phrases from the unstructured text fields of tickets and estimates the frequency for each obtained phrase. The Phrase Refining component applies filters with diverse criteria (e.g., length, frequency, etc.) to refine the extracted phrases.

In the stage of Knowledge Construction, the domain expertise (e.g., the knowledge from system administrators) is utilized for ontology modeling. As usual the ontology is composed of the classes and the relations among classes. The phrases from the Phrase Extracting stage are tagged with the classes defined in the ontology and archived for knowledge base construction. The archived knowledge is leveraged for ticket resolving in the next stage.

The incoming tickets are resolved in the stage of Ticket Resolution. The unstructured text fields of each ticket are first tagged by the Information Inference component. Provided with the tagged ticket, the Recommendation component recommends a ranked list of the most relevant resolutions to the system administrators. The SAs can choose the most appropriate resolution. The ticket with the attached final resolution is archived into the historical ticket repository.

The SAs accumulate more experience during ticket resolution. The newly obtained domain expertise can be used to enrich the knowledge base and facilitate learning. As a result, a closed feedback loop system is formed, and the knowledge base can be incrementally built.

In summary, the enriched knowledge base further facilitates the resolution recommendation, allowing the improvement of IT service management.

III. Design and Implementation

In this section, we explicitly describe the design and implementation for each stage.

A. Phrase Extraction Stage

This stage takes the historical tickets as input and produces useful specific domain phrases (e.g., "available disk space," "backup client connection") by analyzing the unstructured text fields. Intuitively, those phrases encompass the terms with high frequency as well as context information. To achieve this goal, we first extract frequent phrases, then filter out non-informative word combinations to keep only informative phrases. This stage consists of two main components: (1) Phrase Composing and Initial Summary Analysis, and (2) Phrase Refining.

1) Phrase Composing and Initial Summary Analysis: Traditionally, n-gram model is extensively applied to capture the frequently co-occurrent words in a given corpus, explored in our initial approach. However, the extraction of all possible n-grams from a large corpus is an highly time and computing power consuming task. To solve the problem, we exploit the data compression algorithm Lempel-Ziv-Welch (LZW) [4] to extract the hot phrases from the massive ticket corpus.

We address two issues of LZW to achieve our goal of extracting frequent phrases as follows. First, LZW typically works at the character level, and we leverage it to the word level LZW (WLZW). Second, the algorithm only finds repeated patterns but not their frequencies.

Domain-Specific Dictionary Construction: In this part, an input text $\mathcal{T}$ ="*sql server sql server memory*" with repeated patterns is constructed to illustrate how we adopt WLZW for efficient domain-specific dictionary extraction.

Beginning with an empty dictionary, the input $\mathcal{T}$ feeds into the WLZW algorithm. We obtain a dictionary with items (e.g., *"sql," "sql server"*) by reading the first two words. When WLZW reads *"sql"* again, it already exists in the dictionary. Then the algorithm continues to read the next word *"server"* and combine it with the previous word to be a new current phrase *"sql server'* as a key that also exists in the dictionary. Therefore, it keeps reading the next word *"memory"* and merges it with *"sql server,"* a new long phrase *"sql server memory"* composed and inserted into the dictionary.

The WLZW algorithm seeks the trade-off between completeness and efficiency and attempts to find the longest n-gram with a repeated prefix, indicating the importance of the phrase. If an n-gram is not found, it adds the next word and creates an n+1-gram in the dictionary.

The analysis of the time complexity: WLZW runs in a linear time complexity of $O(n)$, where n is the length of the given text. Practically, WLZW takes less than one minute to build the domain-specific dictionary from our entire ticket resolutions.

Frequency of Phrase Estimation: We use the Aho-Corasick algorithm (AC) [5] to locate all occurrences of keys in a dictionary built by the WLZW algorithm and to efficiently calculate the frequency of the found keywords or phrases in the given corpus. The algorithm consists of three parts:

1) Build a Trie (Keyword Tree) based on the domain-specific dictionary,

2) Extend the Trie into a finite state string pattern matching machine to support linear time matching,
3) Fed with the given text, find all matching keywords or phrases appearing as a substring of the input text.

We provides a specific example to clarify how the AC algorithm works in our integrated framework. Assume we have a dictionary $\mathcal{D}$ comprising {*"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"*}. Given the dictionary $\mathcal{D}$, the Aho-Corasick algorithm builds a Trie shown in Fig. 5. The solid arrows are success transitions, while the dashed arrows are failure transitions that might lead to potentially successful matches. If matching the target word, the state of automaton transits in the direction of the arrow from the current state to the following state.

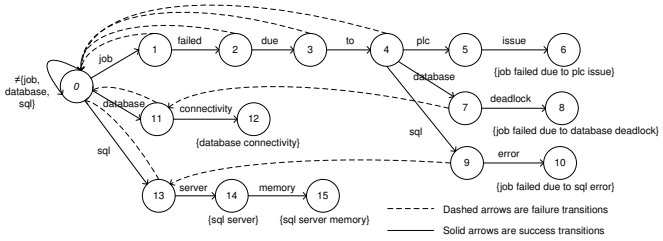


Figure 5: An example of a finite state string pattern matching machine.

We select a real ticket resolution (e.g., *"job failed due to database connectivity"*) as the input, and demonstrate step by step how the AC algorithm finds all matching phrases from the input:

- The automaton stays at the initial state *"State_0"* while scanning non-matching words;
- When reading the word "job," the automaton state transits from *"State_0"* to *"State_1,"* and the output of *"State_1"* is empty;
- Reading word by word, the automaton traverses success transitions (e.g., solid arrows) until it fails in *"State_7;"*
- In *"State_7,"* it transits to *"State_11"* by following a failure transition;
- With the input word "connectivity," automaton transits from *"State_11"* to *"State_12,"* and the output of *"State_12"* is *"database connectivity;"*
- As reaching the end of the word sequence, the matching substring *"database connectivity"* is output.

The analysis of the time complexity: Assume we locate occurrences of a pattern set $\mathcal{P} = \{P_1, P_2, ..., P_k\}$ in text $T[W_1, W_2, ..., W_m]$. Let $n = \sum_{i=1}^{k} |P_i|$ and z is the number of pattern occurrences in T, the AC algorithm runs in a linear time complexity of $O(n + m + z)$.

2) Phrase Refining: The repeated phrases have been extracted during the previous stage; however, not all of the word combinations are useful and some should be omitted from the constructed ontology. Intuitively, we should select the most frequent pattern as important. However, many of them are non-informative phrases (e.g. numbers, "no action"). We apply the following three filters to the extracted repeated phrases allowing the omission of non-informative phrases.

Phrase Length & Frequency Filters: Intuitively, both length and frequency are good indicators for important phrases. Based on our experiments, we define several filtering rules for phrase length & frequent filters: (1) Length ≥ 10 characters; (2) Frequency ≥ 5; (3) Single-word phrases (part of a bi-gram or tri-gram); (4) containing only numbers (non-informative phrases).

With respect to the length threshold setting for Phrase Length Filter, Fig. 6 shows that most of the useful phrases can be obtained when the length falls between 10 and 60. In practice, we keep the phrases longer than 60 as well since those long phrases indicate high frequent occurrences in the WLZW algorithm. The frequency threshold setting is validated by the system administrators considering the trade-off that lower frequency threshold can capture more informative phrases but more noises are included, while higher frequency threshold results in fewer informative phrases.

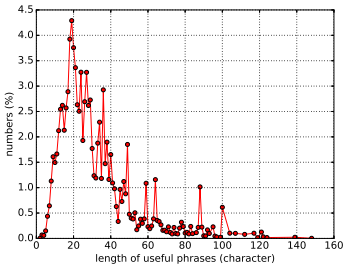


Figure 6: Distribution of length of useful phrases.

Part-Of-Speech Filter: In [6], Justeson et al. claim that technical terms consist mostly of noun phrases containing adjectives, nouns, and occasionally prepositions. They analyze four major technical dictionaries. Subsequently, they come up with seven practical patterns defining a technical term scheme. The scheme and the corresponding Penn Treebank tagset are summarized in Table I. We utilize the existing Stanford Log-linear Part-Of-Speech Tagger [7] to tag input phrases. However, technical terms alone cannot cover all possible informative phrases since our dictionary describes both terms and possible actions (actions may be found in the summary part of the ticket as well as in the resolution part of the ticket). We extend the work [6] by including action describing domain-specific phrases - the phrases that contain verbs in different forms (e.g. past tense verb, gerund, etc.). Corresponding Penn Treebank Action Patterns are outlined in Table II.

Table I: Definition of technical term's schemes.

Justeson-Katz Patterns	Penn Treebank Entity Patterns	Examples in Tickets
A N	JJ NN[P|S|PS]*	global merchant
N N	NN[P|S|PS]* NN[P|S|PS]*	database deadlock
A A N	JJ JJ NN[P|S|PS]*	available physical memory
A N N	JJ NN[P|S|PS] NN[P|S|PS]	backup client connection
N A N	NN[P|S|PS] JJ NN[P|S|PS]	load balancing activity
N N N	NN[P|S|PS] NN[P|S|PS] NN[P|S|PS]	socket connectivity error
N P N	NN[P|S|PS] IN NN[P|S|PS]	failures at sfdc
A:Adjective, **N**: Noun, **P**: Preposition		
JJ: Adjective, **NN**: singular Noun, **NNS**: plural Noun, **NNP**: singular proper Noun, **NNPS**: plural proper Noun, **IN**: Preposition		

The input phrases that do not match defined patterns are eliminated.

Table II: Definition of action term's schemes.

Penn Treebank Action Patterns	Examples in Tickets
VB[D|G|N]*	run/check, updated/corrected affecting/circumventing, given/taken
VB: base form Verb, **VBD**: past tense Verb, **VBG**: gerund Verb,**VBN**: past participle Verb,	

After applying the three filters in a pipeline, a list of candidate phrases, including entities and actions, are created for the class tagging procedure. It provides us a great benefit by reducing unqualified and unmatched potential phrases from manually unmanageable 400+K phrases to approximately 2K candidate phrases. The potential dictionary candidate phrases are ready for manual look-up by domain experts.

B. Knowledge Construction Stage

In the stage of Knowledge Construction, the SAs first develop an ontology model. This ontology model provides the semantic definition of the informative domain-specific phrases obtained during the Phrase Extracting stage.

Second, the phrases with more specific definition are tagged with the classes defined in the ontology, and finally archived for knowledge base construction. To give a concrete example, we are looking for the phrase "database deadlock" instead of just "database," since the former has more specific meaning. The archived knowledge is leveraged for the ticket resolution recommendation in the next stage.

1) Ontology Model: An ontology explicitly defines a common vocabulary including the formal specifications of the terms in the domain as well as the relations among them. Development of an ontology includes [8]:

1) Defining classes in the ontology.
2) Arranging the classes in taxonomic hierarchy.
3) Defining relations amongst the classes.

Then we can construct a knowledge base by defining the instances of these classes (or facts). We build an ontology model with the help of domain experts. To verify coverage and identify capability of our ontology model, we discuss with domain experts the practical situations found in tickets and describe them in terms of the ontology's classes and relations.

Classes: A class is a deterministic concept describing a collection of objects in a given domain [8]. In our ontology model, six classes are explicitly defined in Table III to classify the important domain-specific phrases from previous stages. For example, Entity class represents all technical terms (e.g., memory fault, filesystem error). ProblemCondition class is the description of the negative state of an entity (e.g., stopped, failed).

Relations: A relation describes the interaction among the classes in our ontology model [8]. For example, the Action class can have the "TAKEN ON" interaction on Entity class, and the SupportTeam class can "WORK ON" Entity class. Note that there is no relation between Action class and Activity class. The outline of our ontology models is depicted in Fig. 7.

Table III: Classes of our ontology model.

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

2) Knowledge Archive: Based on our ontology model, a domain expert manually tags the important keywords or phrases with their most relevant classes defined in Section III-B1. For example, the text snippet "certificates will be renewed" can be tagged with classes into tuples such as "[(certificates, Entity), (will, STOP WORD[17]), (be, STOP WORD), (renewed, Action)]." Finally, we initiate our domain knowledge base with approximately 630 instances of Entity class, 240 instances of Activity class, 25 instances of Action class, 21 instances of ProblemCondition class, two instances of Incident class, and 76 instances of SupportTeam.

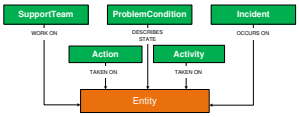


Figure 7: Ontology model depicting interactions amongst classes.

C. Ticket Resolution Stage

The goal of this stage is to recommend operational phrases for an incoming ticket. The incoming ticket is first processed by the Class Tagger module of Information Inference component. Taking the tagged ticket as an input, the Recommendation component provides the list of the most relevant resolutions. Finally, SAs check the recommended results. The ticket is archived into the historical ticket database, and the newly obtained domain expertise can be used to enrich the knowledge base.

1) Information Inference Component: The Information Inference component is used to infer problems, activities, and actions from trouble tickets by applying the constructed knowledge base and ontology model. The three key questions addressed herein are as follows: (1) how to formalize the physical words using the ontology model, (2) how to define three key concepts (e.g., problem, activity, and action) that can be extracted from the tagged ticket, (3) how to find the corresponding entity phrases associated with problem, activity or action phrases.

We address the questions as follows:

Class Tagger Module: The Class Tagger module is an index tool based on our domain knowledge base. Taking the ticket resolutions and knowledge base as the input, it outputs tagged domain keywords or phrases with the corresponding classes. The module has three steps for tagging: (1) tokenize the input into sentences; (2) construct a Trie by using ontology domain dictionary; (3) find the longest matching phrases of each sentence using the Trie and knowledge base, then map them onto the corresponding ontology classes.

For example, *"database," "deadlock,"* and *"database deadlock"* are all valid domain phrases of Entity class. But the Class Tagger module only tags the *"database deadlock"* as Entity in a given sentence. An example of tagged ticket by the Class Tagger module is shown in Fig. 8.

(post loading)/(Entity) (failed)/(ProblemCondition) due to (plc issue)/(Entity). (updated)/(Activity) the (gft)/(Entity) after (proper validation)/(Entity) and (processed)/(Activity) the (job)/(Entity) and (completed)/(Action) successfully.

Figure 8: Ticket tagged by the Class Tagger module.

Defined Concept Patterns for Inference: We first define three key concepts as follows:

Problem describes an entity in negative condition or state.
Activity denotes the diagnostic steps on an entity.
Action represents the fixing operation on an entity.

Using Class Tagger we obtain a total of 672+K tagged ticket resolutions and find some concept patterns in the structured corpus. For instance, ProblemCondition/Action keywords and their corresponding entities always appear in a single sentence. The structure of concepts is identified manually as shown in Table IV.

Table IV: Defined concept patterns for inference.

Concept	Pattern	Examples
Problem	Entity preceded/succeeded by ProblemCondition	(jvm) is (down)
Activity	Entity preceded/succeeded by Activity	(check) the (gft record count)
Action	Entity preceded/succeeded by Action	(restart) the (database)

Problem, Activity, and Action Extraction: The derived concepts provide their patterns for information inference extraction. First, the Class Tagger module tokenizes the input into sentences and outputs a list of tagged phrases. Second, we decide whether it is an informative snippet or not by checking if it exists in a ProblemCondition/Action list. Once ProblemConditon/Action phrase is matched in the sentence, the phrase is appended to the dictionary as a key, and all its related entities are added as the corresponding values via a neighborhood search. Each of the three key concepts has its own dictionary. Finally, we obtain the problem, activity, and action inferences. For instance, given the tagged snippet in Fig. 8, the output is as follows:

- Problem - {failed: plc issue, post loading}
- Activity - {update: gft, proper validation; process: job}
- Action - {complete: job}

2) Ontology-based Resolution Recommendation Component: In our prior work [9] for automatic problem resolution, we propose a KNN-based algorithm in which the resolutions of historical tickets with top summary similarity scores to the incoming ticket summary are recommended. We use the Jaccard similarity function [10] to calculate the summary similarity score after tokenizing each summary into a bag of words.

Typically, Jaccard similarity function ignores the semantic information on ticket summaries. In our application, the ticket summary and resolution are highly noisy, which makes the Jaccard similarity function inappropriate. Table V shows two ticket summaries describing the same issue "database save failed." However, a low Jaccard similarity score here is due to many non-informative words.

Two extended works [11], [12] adopt several techniques trying to alleviate the issue by grouping words into semantic topics or mapping semantically similar words closely in the same vector space. Those approaches, however, only deal with semantically similar words without handling the noise caused by the non-informative words. Fortunately, the ontology model we constructed greatly facilitates our resolution recommendation task, as it essentially enhances our semantic understanding of the tickets and de-noises tickets by filtering the non-informative words out of the textual attributes. De-noising improves similarity allowing tokenized Jaccard similarity function to concentrate only on informative phrases.

Table V: Noisy ticket summary examples.

Inside ProcessTransaction. DetermineOutcome failed. Database save failed: Tried an insert, then tried an update
CRPE3I1Server Database save failed on lppwa899 00:19:46 lppwa899 /logs/websphere/wsfpp1lppwa 899CRPE3I1Server/SystemOut.log [3/20/14 0:19:33:371 MST] 0000002b SystemOut 20140320 00:19:33, 371 [WebContainer:30] [STANDARD] [DI_US:01.22] (ng.AEXP_US_ISR_Work_Txn.Action) FATAL lpp-wa899—10.16.4.4—SOAP—AEXP_US_ISR_Roads3_Pkg —AEXPUSISRWorkInquiry—ProcessInquiry

3) Ontology Construction in ticket summary: Ontology construction in ticket summary follows the same steps as in ticket resolution. But ticket summary delivers the problem symptoms instead of the problem resolution information. It is reasonable to assume that only problem and activity phrases present in ticket summary. Extracted activity phrases describe automatically triggered system actions such as "rerun," "restart," and so on. According to the assumption, only three types of knowledge phrases, i.e., Entity, Activity and Problem Condition, are recognized during the manual tag process.

4) Tokenized Similarity function: Once we extract problems from ticket summary using concept patterns of Table IV, the Jaccard similarity function is applied to the extracted Problem phrases. After removing the non-informative phrases in ticket summary from the process of similarity calculation, the same methodology is adopted for ticket resolution recommendation as in the work [9]. A case study given in Section IV illustrates that the revised similarity function can better capture the similarity between ticket summaries.

IV. Experiment

In this section, we present the dataset, the running environment, and the discussion of experimental results.

A. Data and Setup

Experimental tickets are collected from real production servers of the IBM Tivoli Monitoring system [14]. The data set covers three month time period containing $|\mathcal{D}| = 22,423$ tickets with 33 attributes corresponding to the columns of tickets table.

Our integrated system is designed to compliment monitoring systems such as the IBM Tivoli Monitoring system and to automate delivery of an IT service management. The component is implemented in Java 1.8, and tested on 64-bit Windows 8.1 Enterprise residing on a machine equipped with Intel Core 2 Xeon CPU 3.4GHz and 16GB of RAM.

B. Evaluation Metrics and Evaluation Overview

Four commonly used evaluation metrics are applied in our evaluation. Let TP, TN, FP, and FN correspond to true positive, true negative, false positive, and false negative, respectively. Accuracy is computed as $\frac{TP+TN}{TP+TN+FP+FN}$. Precision is defined as $\frac{TP}{TP+FP}$, recall is defined as $\frac{TP}{TP+FN}$. The F1 score is computed as $2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$.

To evaluate our integrated system, we randomly split our dataset into training and testing dataset. The training set, 90% of the entire ticket dataset, is used to build the knowledge base through our system, while the remaining are used for testing. To build the ground truth, domain experts manually find and tag all phrase instances into six classes defined in Table III. Class Tagger is applied to the testing tickets to produce tagged phrases with predefined classes. Comparing the tagged phrases with the ground truth, we obtain the performance evaluation shown in Fig. 9.

The precision, recall, F1 score, and accuracy for ProblemCondition are close to 1 due to the small number of instances (e.g., failed, occurred, expired, unavailable, etc.). We also observe the precision of Entity, Action, and Activity extraction is 99.86%, 94.42%, and 97%, recall is 88.73%, 95.12%, and 93%, F1 score is 93.97%, 94.77%, and 95.1%, and accuracy is 97.05%, 97.72%, and 99.3%, respectively. The reason is that the classes of Entity, Action, and Activity

contain a large amount of instances in typos and various verb forms. The Incident class is observed with similar results with ProblemCondition class, though its performance is not illustrated explicitly herein.

C. Evaluating Information Inference

We also evaluate the usability and readability of our automated information inference results and compare them with traditional methods of manually analyzed tickets.

For the usability, we evaluate the extracting accuracy for concepts, i.e., Problem, Activity, and Action. Similarly, we tag the ground truth from the testing tickets and then compare it with the result tagged by Information Inference component. We evaluate the average accuracy to be 95.5%, 92.3%, and 86.2% for Problem, Activity, and Action respectively.

To evaluate readability, we focus on measuring the time-cost difference to understand a ticket with and without the Information Inference component. First, 50 tickets are randomly selected from the testing tickets and two domain experts are invited for the task of Problem, Activity, and Action identification. Then, one domain expert is required to execute the task by inspecting these tickets directly, while the other domain expert is presented with the same task utilizing the output from the Information Inference component. We observe a significant decrease in time cost to accomplish the task from around 1000s to 100s totally.

D. Case Study: Resolution Recommendation Task

In this section, we describe a case study of our experimental results and provide the insights learned during the domain experts' manual review process.

For the accurate evaluation one needs to fully understand the semantics of the ticket summary and resolution. That's the reason why the manual review of the recommended results by domain experts is conducted.

The recommendation is achieved based on the similarity score which can be computed by both the word level and the problem level Jaccard similarity functions shown in Table VI. The word level Jaccard similarity function takes the whole textual value of the ticket summary into account for similarity score computing, while the problem level Jaccard similarity function, utilizing the knowledge base constructed in our work, takes only the Problem phrases into account to obtain the similarity score.

To illustrate the difference between the two similarity functions, our task is to recommend the resolution for the ticket with summary *"Patrol Agent is not running,"* which indicates Problem *"not running: patrol agent."* As a fact confirmed by domain experts, Problem *"not running: patrol agent"* is the same as Problem *"offline: patroal agent"* occurring in **B2**, but different from Problem *"not running: zpdc process"* associated with **A2**. However, shown by Table VI, the recommended result based on the word level Jaccard similarity contradicts with the fact. By contrast, the recommended result according to the problem level Jaccard similarity presents the consistency with the domain expertise.

By further investigating our case study, since the entity "patrol agent" mismatches "zpdc process," domain experts assert that the resolution for the later problem contributes little to resolve the previous one. However, if the two entities are similar, such as "zpdc process" and "syslogd process," in the perspective of concept, the resolutions for the entity "zpdc process" might also apply to the entity "syslogd process."

V. Related Work

In this section, we provide a short survey of the literature related to the automated IT service management and knowledge base construction.

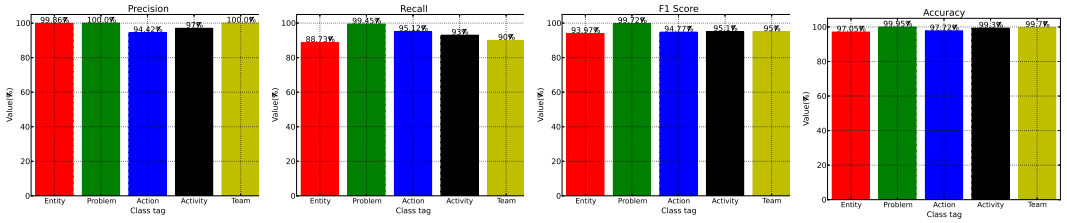
The automation of IT service management is largely achieved through service-providing facilities in combination with automation of subroutine procedures such as problem detection, determination, and resolution for the service infrastructure. Automatic problem detection is typically realized by system monitoring software, such as IBM Tivoli Monitoring [14] and HP OpenView [16]. In [18], Xu et al. develop the automated system runtime problem detection by analyzing console logs. Tang et al. [19] propose an integrated framework to minimize the false positive and maximize the coverage for system fault detection. For problem determination, significant efforts have been put on analyzing structured logs or unstructured text fields. A hierarchical multi-label classification method [20] is proposed to determine the problem types in the monitoring IT tickets. Rish et al. [22] use probabilistic reasoning techniques to solve real-time problem diagnosis in a large distributed system. Automated ticket resolution [9] is a big challenge in IT service management since it requires vast domain knowledge about the target infrastructure. This requirement also inspires our work in this paper. Some prior works apply the approaches in text mining to explore hidden semantic relationship between terms [11], [23], [13]. Another area of interests focuses on the analysis of time series and event data. For example, Zeng et al. [2] adopt an advanced mining algorithm capable of finding fluctuating event correlations and root causes from system failure logs.

However, these studies mainly work on structured data, ignoring valuable domain-specific knowledge hidden in those unstructured text fields/logs. The knowledge base built in our work is not only fundamental to the understanding of the system problems but also can greatly benefit those aforementioned tasks.

Ontology modeling [8] represents domain knowledge that specifies the classes and relations among the classes. It has been extensively investigated by many researchers due to its effectiveness and simplicity, and applied into various research domains (e.g. knowledge management, natural language processing [24], recommender system [25], and so on).

After ontology modeling, the great challenge lies in the knowledge base construction. The authors in [26] analyze natural structured English text to construct the knowledge base. In [27], the authors propose a framework to incrementally build, maintain, and use knowledge bases from Wikipedia semi-structured articles. Lee et al. [28] adopt an episode-based ontology construction mechanism to extract domain knowledge from text documents. However, these studies build their ontology models by taking natural language text as an input.

Our work focuses on mining domain-specific phrases from unstructured texts with little syntax structure and mapping them onto predefined domain knowledge classes to facilitate ontology construction.



(a) Precision. (b) Recall. (c) F1-Score. (d) Accuracy.

Figure 9: Evaluation of our integrated system.

Table VI: Case study for testing ticket summary "Patrol Agent is not running".

Similarity function	Top most similar summary	Associated resolutions
word level	**A1:** Patrol Agent is not running. Problem - {not running: patrol agent}	Server's uptime indicates server was unavailable. Server is available now and patrol agent connectivity present
	A2: The zpdc process is not running Problem - {not running: zpdc process}	Downstream of DB crash
	A3: The syslogd process is not running Problem - {not running: syslogd process}	No actions taken, the process is running as expected on server according to System Operations Procedures
problem level	**B1:** Patrol Agent is not running Problem - {not running: patroal agent}	Server's uptime indicates server was unavailable. Server is available now, patrol agent connectivity present
	B2: Patrol Agent Offline: Failed to reconnect to Patrol Agent on host WWPP, port 3181. Will retry in 3 timer ticks. Problem - {offline: patroal agent}	Verified connectivity. Patrol Agent connectivity test failed.
	B3: The zpdc process is not running Problem - {not running: zpdc process}	Downstream of DB crash

VI. Conclusion

In this paper, we study the problem of constructing a domain specific knowledge base using a large number of tickets in an IT service management system. An integrated cognitive computing framework proposed in our work supports incremental knowledge extraction and ontology construction. We first address the issues of efficient extraction and identification of the domain specific phrases from noisy unstructured text fields in tickets and then construct the knowledge base with the help of domain experts.

We conduct an empirical study that leverages a constructed knowledge base to generate ticket resolution recommendations. Our encouraging results show the effectiveness and efficiency of our integrated framework as applied to the task, and also the scalability to other critical tasks in IT service management system.

There are several avenues for future research. First, we plan to investigate intelligent techniques that reduce the efforts in manual phrase tagging, such as training a conditional random field model [29] to directly tag phrases supervised by an existing knowledge base. Second, our current resolution recommendation task uses problem level Jaccard similarity, which can be further improved by considering the term similarity based on the constructed domain specific ontology [30], [31]. Finally, we plan to incorporate the obtained knowledge base into other tasks in the IT service management system.

VII. Acknowledgments

The work was supported in part by the National Science Foundation under Grant Nos. IIS-1213026 and CNS-1461926, and a FIU Dissertation Year Fellowship.

References

[1] "Itil: Information Technology Infrastructure Library," http://www.itlibrary.org/.
[2] C. Zeng, T. Li, L. Shwartz, and G. Y. Grabarnik, "Hierarchical multi-label classification over ticket data using contextual loss," in *2014 IEEE NOMS*. IEEE, 2014, pp. 1–8.
[3] C. Bartsch, L. Shwartz, C. Ward, G. Y. Grabarnik, and M. Buco, "Decomposition of IT service processes and alternative service identification using ontologies," in *NOMS*. IEEE, 2008, pp. 714–717.
[4] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, 1984.
[5] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.
[6] J. S. Justeson and S. M. Katz, "Technical terminology: some linguistic properties and an algorithm for identification in text," *Natural language engineering*, vol. 1, no. 01, pp. 9–27, 1995.
[7] K. Toutanova and C. D. Manning, "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger," in *SIGDAT*. ACL, 2000, pp. 63–70.
[8] N. F. Noy, D. L. McGuinness *et al.*, "Ontology development 101: A guide to creating your first ontology," 2001.
[9] L. Tang, T. Li, L. Shwartz, and G. Y. Grabarnik, "Recommending resolutions for problems identified by monitoring," in *IFIP/IEEE IM*. IEEE, 2013, pp. 134–142.
[10] G. Salton and M. J. McGill, "Introduction to modern information retrieval," 1986.
[11] W. Zhou, L. Tang, T. Li, L. Shwartz, and G. Y. Grabarnik, "Resolution recommendation for event tickets in service management," in *IFIP/IEEE IM*. IEEE, 2015, pp. 287–295.
[12] W. Zhou, T. Li, L. Shwartz, and G. Y. Grabarnik, "Recommending ticket resolution using feature adaptation," in *CNSM*. IEEE, 2015, pp. 15–21.
[13] W. Zhou, W. Xue, R. Baral, Q. Wang, C. Zeng, T. Li, J. Xu, Z. Liu, L. Shwartz, and G. Y. Grabarnik, "STAR: A System for Ticket Analysis and Resolution," in *SIGKDD*. ACM, 2017, in press.
[14] "IBM Tivoli: Integrated Service Management," http://ibm.com/software/tivoli/.
[15] "IBM Cognitive Computing," http://research.ibm.com/cognitive-computing/.
[16] "HP OpenView: Network and Systems Management Products," http://www8.hp.com/us/en/software/enterprise-software.html.
[17] "NLP: Stop Words," https://en.wikipedia.org/wiki/Stop_words.
[18] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the 22nd ACM SIGOPS*. ACM, 2009, pp. 117–132.
[19] L. Tang, T. Li, L. Shwartz, F. Pinel, and G. Y. Grabarnik, "An integrated framework for optimizing automatic monitoring systems in large IT infrastructures," in *SIGKDD*. ACM, 2013, pp. 1249–1257.
[20] C. Zeng, L. Tang, T. Li, L. Shwartz, and G. Y. Grabarnik, "Mining temporal lag from fluctuating events for correlation and root cause analysis," in *10th International Conference on Network and Service Management (CNSM) and Workshop*. IEEE, 2014, pp. 19–27.
[21] T. Li, C. Zeng, Y. jiang, W. Zhou, L. Tang, Z. Liu and Y. Huang, "Data-driven Techniques in Computing System Management," in *ACM Computing Surveys*. ACM, 2017, in press.
[22] M. Brodie, I. Rish, S. Ma, G. Y. Grabarnik, and N. Odintsova, "Active probing," *IBM TJ Watson Research, Tech. Rep*, 2002.
[23] W. Zhou, T. Li, L. Shwartz, and G. Y. Grabarnik, "Recommending ticket resolution using feature adaptation," in *CNSM*. IEEE, 2015, pp. 15–21.
[24] K. Mahesh, S. Nirenburg *et al.*, "A situated ontology for practical nlp," in *IJCAI*, vol. 19. Citeseer, 1995, p. 21.
[25] W. IJntema, F. Goossen, F. Frasincar, and F. Hogenboom, "Ontology-based news recommendation," in *EDBT/ICDT*. ACM, 2010, p. 16.
[26] M. Y. Dahab, H. A. Hassan, and A. Rafea, "Textontoex: Automatic ontology construction from natural english text," *Expert Systems with Applications*, vol. 34, no. 2, pp. 1474–1480, 2008.
[27] O. Deshpande, D. S. Lamba, M. Tourn, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, and A. Doan, "Building, maintaining, and using knowledge bases: a report from the trenches," in *Proceedings SIGMOD*. ACM, 2013, pp. 1209–1220.
[28] C.-S. Lee, Y.-F. Kao, Y.-H. Kuo, and M.-H. Wang, "Automated ontology construction for unstructured text documents," *Data & Knowledge Engineering*, vol. 60, no. 3, pp. 547–566, 2007.
[29] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the 8-th ICML*, vol. 1, 2001, pp. 282–289.
[30] P. Li, H. Wang, K. Q. Zhu, Z. Wang, and X. Wu, "Computing term similarity by large probabilistic ISA knowledge," in *22nd ACM KDD*. ACM, 2013, pp. 1401–1410.
[31] Z. Wang, K. Zhao, H. Wang, X. Meng, and J.-R. Wen, "Query understanding through knowledge-based conceptualization," in *Proceedings of the 24th ICAI*. AAAI Press, 2015, pp. 3264–3270.